ABSTRACT

This is the teacher's guide and commentary for the SMSG textbook Algorithms, Computation and Mathematics (Algol Supplement). This teacher's commentary provides background information for the teacher, suggestions for activities found in the student's Algol Supplement, and answers to exercises and activities. The course is designed for high school students in grades 11 and 12. Access to a computer is highly recommended. (RH)

ALGORITHMS, COMPUTATION AND MATHEMATICS

(Algol Supplement)

Teacher's Commentary

Revised Edition

The following is a list of all those who participated in the preparation of this volume:

Sylvia Charp, Dobbins Technical High School, Philadelphia, Pennsylvania
Alexandra Forsythe, Gunn High School, Palo Alto, California
Bernard A. Galler, University of Michigan, Ann Arbor, Michigan
John G. Herriot, Stanford University, California
Walter Hoffmann, Wayne State University, Detroit, Michigan
Thomas E. Hull, University of Toronto, Toronto, Ontario, Canada
Thomas A. Keenan, University of Rochester, Rochester, New York
Robert E. Monroe, Wayne State University, Detroit, Michigan
Silvio O. Navarro, University of Kentucky, Lexington, Kentucky
Elliott I. Organick, University of Houston, Houston, Texas
Jesse Peckenham, Oakland Unified School District, Oakland, California
George A. Robinson, Argonne National Laboratory, Argonne, Illinois
Phillip M. Sherman, Bell Telephone Laboratories, Murray Hill, New Jersey
Robert E. Smith, Control Data Corporation, St. Paul, Minnesota
Warren Stenberg, University of Minnesota, Minneapolis, Minnesota
Harley Tillitt, U. S. Naval Ordnance Test Station, China Lake, California
Lyneve Waldrop, Newton South High School, Newton, Massachusetts

The following were the principal consultants:

George E. Forsythe, Stanford University, California
Bernard A. Galler, University of Michigan, Ann Arbor, Michigan
Wallace Givens, Argonne National Laboratory, Argonne, Illinois

Financial support for the School Mathematics Study Group has been provided by the National Science Foundation.

TABLE OF CONTENTS

Chapter TA2

INPUT-OUTPUT AND ASSIGNMENT STATEMENTS

Summary of Chapter A2

Enough of the ALGOL language is introduced in this chapter to enable a student to write very simple programs. One series of exercises is arranged in such a way as to build up complete programs, any or all of which can be computer tested. These are exercises 1-6 at the end of sections

A2-3 Set A
A2-3 Set B
and A2-7.

In addition, Exercise 8, Section A2-7, is also recommended for computer testing. It will be interesting to the better students. You may need to give the students some special help with this one.

The outline:

A2-1	Some background on What is ALGOL and what ALGOL programs look like.
A2-2	The elements of the language, its characters, numerals for constants, variables, labels, names for functions, operators, and special symbols.
A2-3	The read and print statements are introduced. Since ALGOL does not specify format codes for describing input and output records, only a general awareness of the problem is transmitted to the students. A suggested free form for data cards and a standard formatted output are imagined. These are explained by discussing the execution of read and print procedures in a typical ALGOL implementation.
A2-4	The assignment statement is explained largely in terms of what has been learned from the extensive material in the flow chart text. The student should pay attention to the difference between real and integer division, the latter being explained in terms of the greatest integer function.
A2-5	Order of computation in an ALGOL expression is explained in terms of the material given in the flow chart text.
A2-6	Converting integers to reals and vice versa is shown to be accomplished by the assignment statement.

A2-7 A simple (but complete) ALGOL program is displayed and a set of exercises given where the student is asked to write his first complete programs. The concept of "head" and "body" of a program is introduced. So is the notion of a compound statement as opposed to a simple statement. The concept of a "block" is not mentioned, however.

Literature on ALGOL 60 and its implementation

A nonexhaustive list is provided here.

A. Reference manuals (The manuals listed are revised frequently and the document numbers that are given may not reflect the latest revision.)

		Computer
1.	"Extended ALGOL Reference Manual for the B5000." Burroughs Corp., Bull. No. 5000-21012	B5500
2.	"Miscellaneous Facilities of Extended ALGOL for B5000." Burroughs Corp., Bull. No. 5000-21013	
3.	Burroughs Algebraic Compiler, Bull. No. 220-21017	B220
4.	ALCOR Algol for the IBM 7090 Latest reference manual available from the University of Illinois Computer Laboratory	IBM 7090/94

B. Primers, guides and other texts

The best source of such material is the SMSG annotated bibliography entitled "Study Guide in Digital Computing and Related Mathematics," which is reprinted at the end of the Teachers Commentary for the Main Text. See especially the references mentioned in Section III, Algorithmic Languages.

A2-1 "Target" programs and source programs

Terminology changes rapidly in a field which is moving as rapidly as the computer field. Most of the older literature used the term object program. We are using the currently preferred term in choosing the word target.

In addition to the normal manner of processing a source program as described in the student text, there is an alternative approach which is worth knowing about. In this approach, the processor or compiler program produces a target program which is executed in the "interpretive mode."

This type of target program consists of instructions that are not strictly machine codes. They are machine-like instruction codes, often called an "interpretive" code.

In order to execute such a target, a specially developed "interpreter" program must be stored in memory along with the target before execution can proceed. Such compilers have been very successful, especially on machines with limited memory such as the IBM 1620. The interpreter program has the task of interpreting and then carrying out the intent of each pseudo instruction of the target code. The interpreter program in a sense simulates a computer within a computer. The success of these compilers is explained by the fact that an ALGOL source program translated into interpretive code often occupies far less memory (fewer pseudo instructions) than an ALGOL source program which is translated into machine code (more actual instructions). In the former (interpretive) case, the total combined memory requirement for the interpretive code produced by the compiler and the interpreter program is normally less than that for the straight machine code. It is for this reason that this approach is popular for machines of limited memory. On the other hand, the latter (machine code) case normally results in faster-running programs.

Some computers have been designed and built so that the task of compiling is made easy. The Burroughs B-5500 computer is an example. An ALGOL compiler for this computer develops a target program expressed in machine code which is as compact as could be obtained with most interpretive approaches -- so the advantages of both storage economy and running speed are thereby achieved with essentially none of the disadvantages.

A2-2 The Character Set

Many of the characters shown in Table A2-1 are also shown in the card picture, Figure 1-15 in Chapter 1 of the main text.

The special arrangement of letters and digits in Table A2-1 is to emphasize that letters and digits in the same column of the table have one hole punch of their code in common. For example, if you inspect the card picture you will see that the letters E, N, and V and the digit 5 each have a punch in row 5.

Character	Row punches used
E	12, 5
N	11, 5
V	0, 5
5	5

Those characters which are not shown in Figure 1-15 are generally not available on the standard "office printing key punch" which is usually the IBM Model 026. Special models are being built, not always available, which have many of the special characters needed in ALGOL programs, like

: ; ↑

Notice that even with a special key punch we still lack the <u>lower</u> case letters. Accordingly, when translating a handwritten ALGOL program to punch cards, a number of agreements and compromises must still be made, and even if we could key punch the lower case letters, we would still have a problem in recognizing the <u>underlined</u> or boldfaced words like <u>begin</u>, <u>comment</u>, <u>go</u> <u>to</u> and <u>end</u>.

The significance of these special symbols in ALGOL is explained at the end of Section A2-2. How, then, do computer implementations of ALGOL distinguish between, say

<u>begin</u>, begin, and BEGIN

if all three must be punched on a card in upper case? The answer is they cannot be distinguished. Two approaches to a solution to this difficulty have been used. The first approach requires that some special delimiter character, like the apostrophe, be placed on either side of the word if we mean the special symbol. Thus, <u>begin</u> should appear as 'begin'. In this same approach the identifiers begin and BEGIN in an ALGOL program would both be punched as BEGIN and must therefore have the same meaning.

The second approach is simply to reserve character strings like BEGIN, GO TO, END, REAL, etc. That is, give them one meaning only--namely the special symbols begin, go to, end, real, etc. This means we may not use the same character strings as variables, labels, or function names. The programmer's job, then, is to memorize these 15 or 20 special symbols and be sure never to use these in any other context.

The punched cards which are illustrated in Figure A2-5 were prepared on a special IBM Model 026 key punch to which seven special characters have been added. These are

[ ] ← : < ≤ ;

It has a special punch, the ← ,which is used to signify the ALGOL assignment symbol (:=). In one compiler system the ←, produced by a single key stroke, must be used when we mean the assignment symbol :=, even though both : and = are also part of the set. This is because := in ALGOL is thought of as a single character and not a pair of characters. Figure TA2-1 shows the full character set available on the special IBM 026.

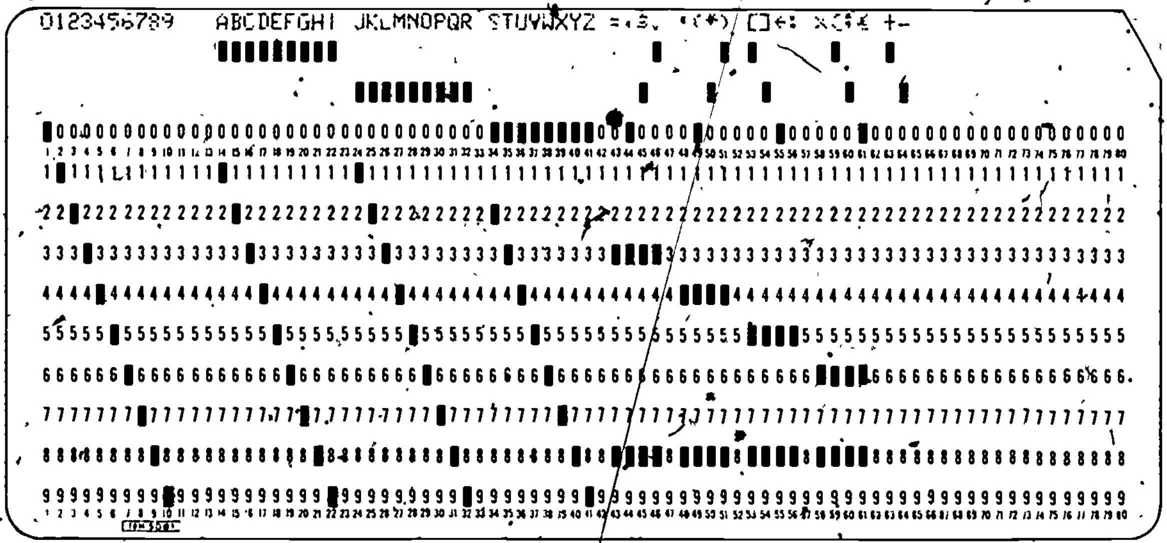

0123456789 ABCDEFGHI JKLMNOPQR STUVWXYZ =.$, '(*)/[]←: <≤;≤ +-

Figure TA2-1. Punch card codes realized on the special IBM Model 026 key punch

Special Characters

Five of the special characters in Figure 1-15 were placed on key punches when ALGOL became available several years ago. Prior to that time other special characters preferred by the business community were used. The business community was and still is the largest user of key punches. So, if you need key punches for your laboratory classes and obtain the use of a "business" key punch, you can expect the keys to display a different set of special characters. There is an equivalence between the ALGOL set and any other set in the sense that up to now all key punch machines punch only one set of hole combinations, i.e., the ones shown on the card picture. Only the characters printed on the keys, and the corresponding characters that print at the very top of the card in each column, may differ. The most typical equivalence is

ALGOL	Business
(	%
)	▱ (lozenge)
+	& (ampersand)
=	@ (at, each)
' (apostrophe)	- (dash)

Interpretation of the semicolon

In the student text we are employing the semicolon as if it were a statement terminator. That is, we are suggesting to the student that the semicolon be used to terminate each statement. Strictly speaking, however, the semicolon is used to separate statements. In certain instances the semicolon is not strictly necessary, especially when a statement is followed by end. Thus in Figure A2-4 the semicolon used in

go to START;

is not necessary. Since in these instances the superfluous semicolon does no harm, we prefer to use it consistently rather than to frequently call the student's attention to the fact that it is unnecessary.

Answers to Exercise A2-2 Set A

Numbers below are constructed from parts shown below:

$-14_{10}{+04}$	1; 2b, 3	where
52.0	2c	1 means sign
-017.14	1, 2c	2b means integral part
10'	3	2c means fractional and integral parts
$+10_{10}{-10}$	1, 2b, 3.	3 means scale factor

Answers to Exercise A2-2 Set B

2JOHN is invalid since the first character is not a letter a - z or A - Z.

M/4
T.6
F-6

are invalid since they each contain a special character that is neither a letter nor a digit.

A2-3 Input-Output Statements

When assigning these exercises, you may wish to suggest that all the students use the same data values. In exercises of Set B of this section, the students may want to compute results for later comparison with computer output to be developed with the exercises in A2-7 Set B. One set of suggested data values is given here.

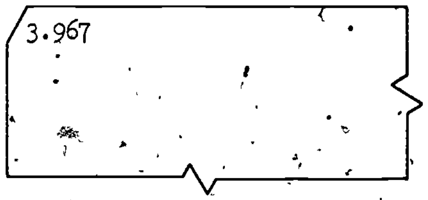
For Exercise 1 use T = 3.967
" 2 " n = 20, i = 3, and j = 4
" 3 " A = 3.0, B = 4.0, C = -2.5, D = 1.5, and X = 2.0
" 4 " m = 12.5 and n = 2.0
" 5 " A = 4.14, Y = 2.01
" 6 " r = 10.0, s = 9.0 and PHI = 1.11977 (arcsine of $\frac{9}{10}$)

The answers below may differ in detail from what is correct in your system. Make the necessary corrections. In particular if you use punch cards, only upper case letters are available. Also some systems use commas to separate numerals on data cards instead of blanks.

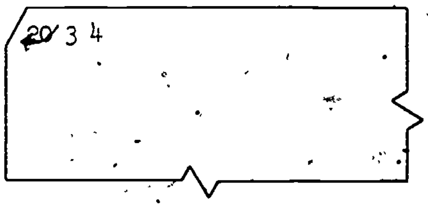
Answers to Exercises A2-3 Set A

read statement — Data card picture

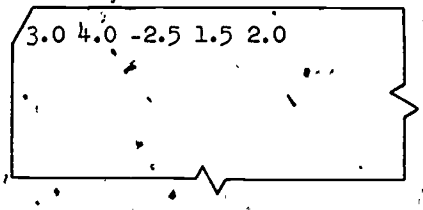
1. read (T);



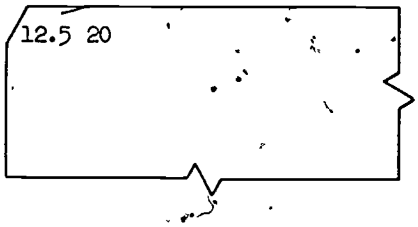
2. read (n, i, j);



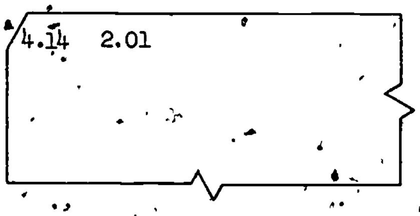
3. read (A, B, C, D, X);



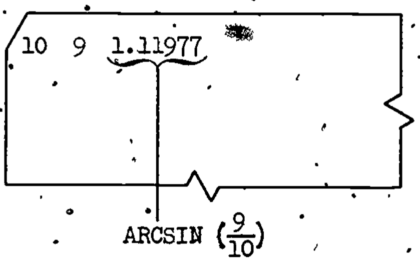
4. read (m, n);



5. read (A, Y);



6. read (r, s, PHI);



Answers to Exercises A2-3 Set B

	write statement	appearance of printed result (actual value)
1	write (Z);	$\boxed{6.4670}$
2	write (ℓ);	$\boxed{44}$
3	write (Z);	$\boxed{36.5000}$
4	write (Q);	$\boxed{4.0000}$
5	write (X);	$\boxed{0.4911}$

.6 write (AREA) $\boxed{5.8726}$

The student need not be asked to develop his results with as many place accuracy as shown here for 5 and .6. Three-place accuracy is probably sufficient in all cases. Correct the second column, "appearance of printed result", to apply to your particular computer.

Answers to Exercise A2-4 Set A

1.

operands		operation	Type of result
A	B	⊙	A ⊙ B
2	3	+	integer
6.3	.09	×	real
100	.4	−	real
6	.4	/	real
6	.4	÷	undefined

2. Verifying the mathematical formula for integer division

Case	A	B	① (A/B)	② sign(A/B)	④[†] abs(A/B)	⑤ entier(abs(A/B))	⑥ A ÷ B
1	9	10	0.9	+1	0.9	0	0
2	-10	-10	1	+1	1	1	1
3	-11	-10	1.1	+1	1.1	1	1
4	10	1	10	+1	10	10	10
5	-5	10	-0.5	-1	0.5	0	0
6	-15	10	-1.5	-1	1.5	1	-1
7	10	-1	-10	-1	10	10	-10
8	1	-10	-0.1	-1	0.1	0	0

[†] Step ③ is left out because it is the same as Step ①.

The formula

$$A \div B = \text{sign}(A/B) \times \text{entier}(\text{abs}(A/B)),$$

may be understood better if the student is urged to carry out the steps in evaluating the right-hand side as if the equal sign were a replacement operator of an assignment step.

The example, for A = 9, B = 10, is shown in the figure below. It might be worthwhile, as an exercise, to require the student to develop a similar figure for this or one of the other seven examples given in the student text.

	Explanation	Actual Value
$9 \div 10 = \text{Sign}(9/10) \times \text{entier}(\text{abs}(\underbrace{9/10}))$	Form R1 = 9/10	0.9
$\underbrace{\text{abs}(\ R1\ )}$	Form R2 = abs(R1)	0.9
$\underbrace{\text{entier}(\ \ R2\ \ )}_{R3}$	Form R3 = entier(R2)	0
$\underbrace{\text{Sign}(\ R4\ )}$	Form R4 = 9/10	0.9
$\underbrace{R5 \quad \times \quad R3}$	Form R5 = Sign(R4)	+1
$9 \div 10 = R6$	Form R6 = R5 × R3	0
	9 ÷ 10 is R6	

<u>Answers to Exercises A2-4, Set B</u>

We have asked the student, as an exercise, to extract a rule for exponentiation out of the twenty-two cases given him.

let i be a number of type <u>integer</u>
let r " " " " " <u>real</u>
let a " " " " either type

The rule may be stated thus:

		Conditions and significance	Type of result
Category 1.	a↑i	If $i > 0$: $a \times a \times \ldots \times a$ (i times)	Same as for a
		If $i = 0$, if $a \neq 0$: 1	Same as for a
		if $a = 0$: undefined	
		If $i < 0$, if $a \neq 0$: $1/(a \times a \times \ldots \times a)$	real
		if $a = 0$: undefined	
Category 2.	a↑r	If $a > 0$: exp(r × ln(a))	real
		If $a = 0$, if $r > 0$: 0.0	real
		if $r \leq 0$: undefined	
		If $a < 0$: always undefined	

Answers to Exercises A2-4 Set C

1. To express $A^{3/2}$

(a) abs(A↑1.5)	abs is unnecessary because A is known to be positive so $A^{3/2}$ must also be positive.
(b) (A↑3)↑0.5	o.k., but expensive computationally (will require log-antilog procedure).
(c) sqrt(A↑3)	This is the best way. (Requires the least amount of computation.)
(d) abs(sqrt(A↑3))	abs function is unnecessary.
(e) (A↑1.5)	o.k., but expensive (requires the log-antilog process).
(f) abs(A)↑1.5	abs function unnecessary. Also, it requires log-antilog process.
(g) sqrt(abs(A↑3))	abs function unnecessary--otherwise, as good as (c).

2. Only (f) or (g) would be satisfactory to express $|A|^{3/2}$. (g) is preferred because it avoids the log-antilog process. Another alternative would be sqrt(abs(A)↑3).

Answers to Exercise A2-4 Set D

In the three incorrect statements below the first and third can be corrected without ambiguity. The Algol in the second statement may be corrected in two ways, but the resulting computations are different.

(1) T := B × -A; should be T := B × (-A);

but

(2) F := C/-3 + 4; may be corrected as F := C/(-3) + 4;
or as F := C/(-3 + 4); These in general will yield different results.

similarly

(3) G := A + B × (C × -F/D); may be corrected
as G := A + B × (C × (-F)/D);
or as G := A + B × (C × (-F/D)); These are computationally equivalent.

Answers to Exercises A2-6

1. (a) 4. is an invalid ALGOL constant. It must be written as 4 or 4.0

(b) * is an invalid ALGOL operator. Should be × . EXP as distinguished from exp is o.k. unless the particular ALGOL implementation you are using cannot distinguish between lower case and capital letters. In this event the expression would be considered invalid since EXP will be interpreted as the standard mathematical function but is not followed by a parenthesized argument expression.

(c) Invalid use of the symbol sin. If sine is meant then there should be an argument enclosed in parentheses following sin.

(d) O.K.

(e) O.K.

2. A := exp(EXP↑2×(A4×EXP + A3));

3. <u>real</u> FPART,V;
FPART := V - entier(V);

4. <u>real</u> V;
<u>integer</u> INTPT;
INTPT := entier(V);

5. They are not the same. The flow chart box that's needed is

$$\longrightarrow \boxed{J \leftarrow \text{ROUND(TRUNK(I/K))}} \longrightarrow$$

But since the TRUNK function is an <u>integer</u> rounding function, TRUNK(I/J) will <u>be</u> an integer. Therefore the ROUND function, also an integer rounding function, will have no further effect.

$$\text{ROUND(TRUNK(I/J))} = \text{TRUNK(I/J)}$$

The flow chart box

$$\longrightarrow \boxed{J \leftarrow \text{TRUNK(I/K)}} \longrightarrow$$

would be a correct answer.

6. They are not the same. The flow chart that's needed is

$$\longrightarrow \boxed{J \leftarrow \text{ROUND(I/K)}} \longrightarrow$$

A2-7 Writing Complete ALGOL Programs

This book makes no attempt to teach <u>everything</u> about ALGOL. In particular, no mention is made in the student's book of <u>block</u> structure. The omission should cause no difficulty. Students are however, instructed to group all declarations together at the beginning of a program. Later when this book has been mastered, the block structure of ALGOL can be pursued further.

Some ALGOL compilers require more declarative information than the ALGOL 60 requires. For example, one ALGOL compiler (Burroughs ALGOL), requires that all statement labels be declared. For the program in Figure A2-10, where the first statement in the "body" is

AGAIN: read (PRICE);

a declaration like <u>LABEL</u> AGAIN;

would be needed in the "head" of the program.

You should for this reason get the help of a local expert who is familiar with the detailed requirements of the particular ALGOL compiler you will be using for your laboratory work. Such a person can save you much time in getting started. You might get him (or her) to take the program in Figure A2-10 and run through all phases of testing with you starting with the key punching and continuing through the satisfactory execution of the program.

Answers to Exercises A2-7

```
1.   begin      real T,Z;
         START:read (T);
                Z := 2.5 + T;
                write (Z);
                go to START;
     end

2.   begin      integer n, i, j, ℓ;
         START:read (n, i, j);
                ℓ := n × (i - 1) + j;
                write (ℓ);
                go to START;
     end

3.   begin      real  A, B, C, D, X, Z;
          HERE:read (A, B, C, D, X);
                Z := ((A × X + B) × X + C) × X + D;
                write (Z);
                go to HERE;
     end

4.   begin      real m, n, Q;
        STAART:read (m, n);
                Q := sqrt((m - 4.5) × n);
                write (Q);
                go to STAART;
     end

5.   begin      real A,Y,X;
        begine:read (A, Y);
                X := 2/(Y + A/Y);
                write (X);
                go to begine;
     end
```

6.
```
begin   real r, s, PHI, rsq, AREA;
     ici:read (r, s, PHI);
        rsq := r × r;
        AREA := 3.14159/2 × rsq
        -(s × sqrt(rsq - [illegible] × s) + rsq × PHI);
        write (AREA);
        go to ici;
end
```

7.
```
NUM20 := entier(PRICE/20.0);
```

8.
```
        ALGOL program
begin   comment the carnival wheel problem;
        integer s, m, k, p;
   START:read (s, m);
        k := m + s -((m + s)÷4) × 4;
        p := 20 × k - 30;
        write (p);
        go to START;
end
```

Chapter TA3

BRANCHING AND SUBSCRIPTED VARIABLES

Introduction.

When the student has mastered the flow chart text for Chapter 3 and this companion ALGOL, he is fully equipped to solve computational problems of essentially any complexity. All the basic programming tools are at his disposal. Much of the material in subsequent chapters falls in the category of important refinements, shorthand techniques, example applications and special concepts. That is why it is very tempting, upon completion of this chapter, to tarry and solve a large number of problems. We avoid this temptation as much as possible because the refinements to be introduced in each succeeding chapter make the solution of problems increasingly easy and interesting. We expose the student to relatively few applications here. Exercises involving algorithms emphasize analysis, mainly, rather than synthesis. The shift to synthesis is a gradual process which is accelerated in Chapter 4.

The two fundamental ideas of Chapter 3, namely branching and subscripted variables, are mirrored rather easily in the Chapter A3 requiring relatively few new ideas.

Indeed, if we were to limit the form of the ALGOL if to simply

```
if relational expression then go to BOX X
                         else go to BOX y;
```

there would be no new idea at all.

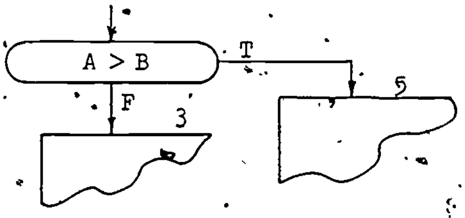
The only new idea associated with subscripted variables in ALGOL is that the programmer has the responsibility and must declare how much memory space is to be allocated for each vector or array that is employed in an ALGOL program. This is done with the array declaration in which one gives the range of each subscript of each vector or array variable.

Outline of Chapter A3

A3-1 The conditional *if* statement is introduced. Examples and exercises are given to show the relationship between flow chart and ALGOL. For example,

if A > B *then go to* BOX5 *else go to* BOX3;

is explained as the equivalent of



The form of the expression between the words *if* and *then* is not elaborated except to tell the student that, like the simple condition box of the flow chart language, it consists of an arithmetic expression, followed by one of the six relational symbols (operators),

$$= \quad \neq \quad < \quad \leq \quad > \quad \geq ,$$

followed by another arithmetic expression.

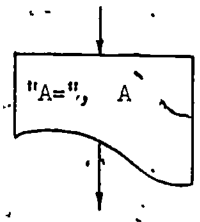
Several variations in the overall form of the statement are then introduced which make coding, i.e., transliterating from flow chart to ALGOL, more convenient.

The basic forms are summarized below:

1. *if* relational expression *then* any simple or compound statement;
2. *if* relational expression *then* any simple or compound statement *else* any statement;

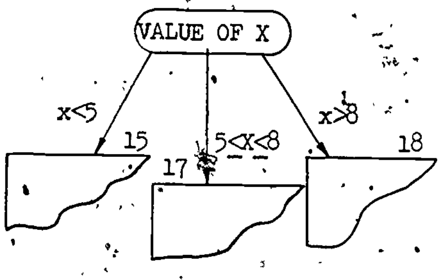
A generalization is made which permits the nesting of one conditional entirely within another. Graded sets of exercises are given to permit the student to gain a gradual appreciation of these various *if* forms. The variations are merely a convenience. Only the basic forms are really essential.

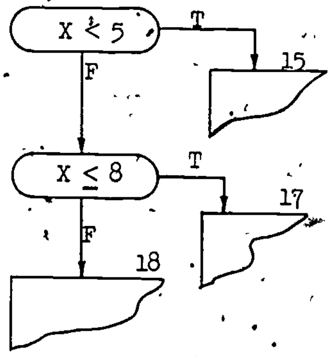
One other topic brought up in this section is the use of literals (items in quotation marks) as output list elements--to correspond to the same usage in the flow chart. Thus, write ("A = ", A); corresponds to



A3-2 Since the flow chart text covers the subject of <u>auxiliary variables</u> in a way that transfers to ALGOL in a straightforward fashion, this section consists only of an example plus some exercises.

A3-3 The transliteration of compound condition boxes from flow charts to ALGOL conditional statements is explained and illustrated. The <u>else if</u> form of the <u>if</u> statement is used to advantage for multi-way branching. For example





which then allows use of the form;

<u>if</u> relational expression <u>then</u> <u>go</u> <u>to</u> BOXi <u>else</u> <u>if</u> relational expression <u>then</u> <u>go</u> <u>to</u> BOXj <u>else</u> <u>if</u> relational expression etc.,

or specifically,

<u>if</u> $X < 5$ <u>then</u> <u>go</u> <u>to</u> BOX15 <u>else</u> <u>if</u> $X \leq 8$ <u>then</u> <u>go</u> <u>to</u> BOX17 <u>else</u> <u>go</u> <u>to</u> BOX18;

A3-4 No new material is included here in paralleling the flow chart text as all the ideas carry over to ALGOL in one-to-one fashion. (Precedence level of relational operators with respect to those for the arithmetic operators.)

A3-5 The ALGOL form for singly-subscripted variables is introduced. Array declarations and their relation to storage allocation are discussed. Finally, a crude array input-output technique is illustrated (to be supplanted by a slightly more elegant technique in Chapter 4).

A3-6 Everything said about singly-subscripted variables in A3-5 is repeated for doubly-subscripted variables. Input-output of entire arrays is left until Chapter 4 where with the use of iteration statements it is made easy to achieve.

The student is asked to work numerous exercises throughout the chapter which involves construction of ALGOL programs from corresponding flow charts presented as examples or developed as exercises in the flow chart text.

Answers to Exercises A3-1 Set A

1. Invalid. Can't have two relational symbols as shown.
2. Invalid. if must be underlined.
3. O.K.
4. Invalid. Can't have semicolon before then.
5. Invalid. Can't have two relational symbols as shown.

Comment: In the more complete ALGOL language, not described here to the student, logical operators like and, not and or are permitted so it would be possible, say, in 1 to write

if A < B and B < C then

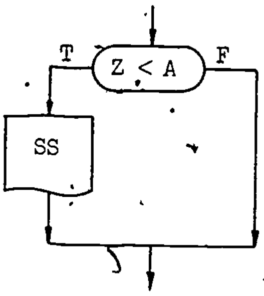
which is a valid if clause having more than one relational symbol. Relational symbols can be thought of as operators and are explained in this sense in Section 3-4.

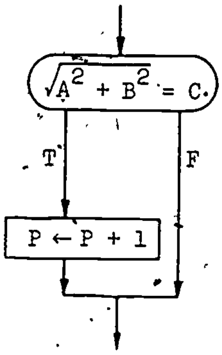
6. <u>if</u> w ≥ 0 <u>then</u> Y := Z + X;

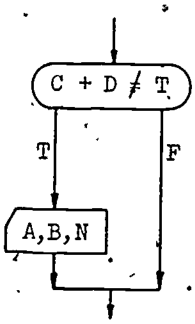
7. <u>if</u> J < 0 <u>then</u> T := 3 × T;

8. <u>if</u> K - 4 = 0 <u>then</u> read(Z);

9. <u>if</u> x > 9.7 <u>then</u> write(Y,T);

10. 

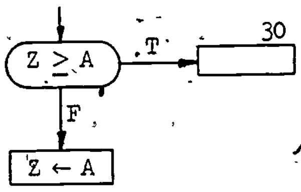
11. 

12. 

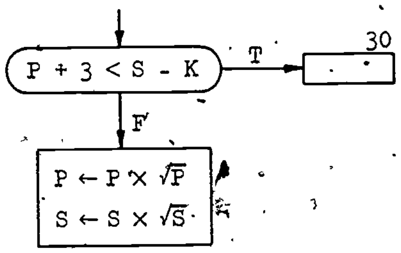
<u>Answers to Exercises A3-1 Set B</u>

```
1.  if I = J then begin F := G + H;
                        P := sqrt(T ↑ 3);
                   end;

2.  if A + B > C then begin Y := Z + X;
                            T := 3 × T;
                       end;

3.  if I = J then go to box2;

4.  if I = J then go to box2;
```

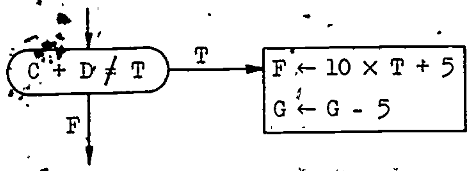
5.



6.



7.



Answers to Exercises A3-1 Set C

```
1.  if  K = 0  then  write ("case 1") else  write ("case 2");
    BOX30:

2.  if  K = 0  then begin  T := 1;
                           write (S);
                     end
               else begin  T := 2;
                           write (SS);
                     end;
    BOX30:

3.  if  S < T  then  F := F - 1
               else  write (Q);
    BOX30:

4.  if  S < T  then  S := S + 1
               else begin  K := K + 1;
                           N := N + 1;
                           S := 0;
                     end;
    BOX30:

5.  if  C + D ≠ T  then go to BOX30
                   else begin  F := 10 × T + 5;
                               G := G - 5;
                               go to BOX40;
                        end;
    BOX30:
```

Comment: Some students may think of negating the conditional of Problem 5 and interchanging the "T" and "F" labels on the exits. This results in an alternate solution:

```
        if C + D = T then begin F := 10 × T + 5;
                                G := G - 5;
                                go to BOX40;
                          end;
        BOX30:
```

Further discussion of this idea has been deferred until later in the book.

6.
```
if A < B then begin T := A;
                    A := B;
                    B := T;
                    go to BOX40;
         end;
BOX30:
```

7.
```
if F + G > H then begin S := S + 1;
                        T := T + 1;
                        U := S × T;
                  end
             else begin P := P + 1;
                        Q := Q + 1;
                        R := P/Q;
                  end;
BOX30:
```

8. Semicolon before else must be removed.

9. Has missing then, or else else has been used when then was intended. A correct version might be

```
        if COUNT < 100 then go to BOX40;
```

10. COUNT + 1 is not a relational expression.

11. No matching end for the begin.

Answers to Exercises A3-1 Set D

```
1.  write ("the impossible");

2.  write ("A□=□", A);

3.  write ("A□=□", A, "□□B□=□", B, "□□C□=□", C);
```

Comment on space symbol "□".

From now on we will omit the space symbol, □. The exact space count is not usually of importance to us here.

Answers to Exercises A3-1 Set E

```
1.  begin real b, c, d, x;
          read (b, c, d, x);
          write (b, c, d, x);
          if b > c  then write (d) else write (x);
    end

2.  begin real b, c, d, t, x;
          read (b, c, d, x);
          write (b, c, d, x);
          if  d < c  then  t := c × b + d × x
                     else  t := d - c;
          write (t);
    end

3.  (a) begin real b, c, d, x, t, u, w, y;
              read (b, c, d, x);
              write (b, c, d, x);
              t := b + c;
              u := b × c × d;
              if  t ↑ 2 + x ↑ 2 > u
                  then begin  w := t + u; write (w); end
                  else begin  w := u ↑ 2;  y := t ↑ 8;  write (w,y); end;
        end
```

```
(b) begin  real b, c, d, x, w, y;
           read (b, c, d, x);
           write (b, c, d, x);
           if (b + c)↑2 + x↑2 > b × c × d
                then begin w := b + c + b × c × d; write (w); end
                else begin w := b↑2 × c↑2 × d↑2;
                           y := (b + c)↑8;
                           write (w, y); end;
    end

4.  begin          integer j, m, n, sum;
          BOX1:    read (j, m, n);
                   if  m > n  then  sum := j + m
                              else  sum := j + n;
                   write (j, m, n, sum);
                   go to  BOX1;
    end

5.  begin          real  b, c, x;
          BOX1:    read (b,c);
                   write (b,c);
                   if  b = 0
                   then go to  BOX4
                   else begin  x := -c/b;
                        write ("The root of  bx + c = 0 is", x);
                        go to  BOX1; end;
          BOX4:    if  c = 0
                   then write ("Every real number satisfies  bx + c = 0.")
                   else write ("bx + c = 0  has no root.");
                   go to  BOX1;
    end
```

A simpler appearing program can be written if we nest the if's and avoid the label BOX4:

```
begin           real b, c, x;
        BOX1:   read (b, c);
                write (b, c);
                if b = 0
                then begin if c = 0 then
                     write ("Every real number satisfies bx + c = 0")
                     else write ("bx + c = 0 has no root"); end
                else begin x := -c/b;
                     write ("The root of bx + c = 0 is", x);
                     end;
                go to BOX1;
end
```

Answers to Exercises A3-1 Set F

```
1.  begin           real SUMALL, T;
                    integer COUNT;
                    COUNT : = 1;
                    SUMALL : = 0;
            BOX2    read (T);
                    SUMALL : = SUMALL + T;
                    COUNT : = COUNT + 1;
                    if COUNT > 100
                    then write ("SUMALL = ", SUMALL)
                    else go to BOX2;
    end

2.  begin           real SUMCUB, T;
                    integer COUNT;
                    SUMCUB := 0;
                    COUNT := 1;
            BOX2:   read (T);
                    SUMCUB := SUMCUB + T ↑ 3;
                    COUNT := COUNT + 1;
                    if COUNT > 100
                       then write ("SUMCUB = ", SUMCUB)
                       else go to BOX2;
    end
```

31

```
3.  begin         real SUMNEG, T;
                  integer COUNT;
                  SUMNEG := 0;
                  COUNT := 1;
            BOX2: read (T);
                  if T < 0 then SUMNEG := SUMNEG + T;
                  COUNT := COUNT + 1;
                  if COUNT > 100
                     then write ("SUMNEG = ", SUMNEG)
                     else go to BOX2;
    end

4.  begin         real SUMALL, SUMCUB, SUMNEG, T;
                  integer COUNT;
                  SUMALL := 0;
                  SUMCUB := 0;
                  SUMNEG := 0;
                  COUNT := 1;
            BOX2: read (T);
                  SUMALL := SUMALL + T;
                  SUMCUB := SUMCUB + T ↑ 3;
                  if T < 0 then SUMNEG := SUMNEG + T;
                  COUNT := COUNT + 1;
                  if COUNT > 100
                     then write ("SUMALL = ", SUMALL, "SUMCUB = ", SUMCUB,
                                 "SUMNEG = ", SUMNEG)
                     else go to BOX2;
    end

5.  begin         real CUMSUM, T;
                  integer COUNT;
                  CUMSUM := 0;
                  COUNT := 1;
            BOX2: read (T);
                  CUMSUM := CUMSUM + T;
                  write ("CUMULATIVE SUM = ", CUMSUM);
                  COUNT := COUNT + 1;
                  if COUNT > 100 then DUMMY:
                     else go to BOX2;
    end
```

Comment: The statement labeled DUMMY is empty. It is a "dummy" statement in the sense that it accomplishes nothing. When COUNT > 100 is true, the dummy statement is executed and control reaches the end of the program. In ALGOL a statement may be represented purely by its LABEL.

To avoid the use of this dummy until the student is introduced to it later, we may replace the if statement with this alternative:

```
if COUNT ≤ 100 then go to BOX2;
```

```
6.  begin     comment Badminton or Volleyball;
              real T;
              integer I, State, ScA,ScB;
              I := 0;
              State := 0;
              ScA := 0;
              ScB := 0;
              BOX2: if I ≠ 100 then
              begin read (T);
                        if T ≥ 0 then
                            begin if State = 0 then
                                ScA := ScA + 1 else State := 0;
                            end
                                else if State = 0 then
                                State := 1 else ScB := ScB + 1;
                        I := I + 1;
                        go to BOX2;
              end
              else if ScA = ScB then write ("TIE GAME", ScA, "ALL")
                  else if ScA > ScB then
                          write ("PLAYER A WINS", ScA, "TO", ScB)
                      else write ("PLAYER B WINS", ScB, "TO", ScA);
    end
```

Answers to Exercises A3-2, Set A

```
1.  begin          comment Fibonacci sequence to produce Random Numbers;
                   integer  LTERM, NLT, I, COPY;
                   LTERM := 1;
                   NLT := 0;
                   I := 1;
            BOX2:  if  I < 117  then
                   begin  BOX3:  COPY := LTERM;
                                 LTERM := LTERM+NLT-1000×entier((LTERM+NLT)/1000);
                                 NLT := COPY;
                                 I := I + 1;
                                 if  I < 17  then go to BOX3
                                     else begin write(I, LTERM);
                                          go to BOX2;
                                          end;
                   end
                   else HALT:
    end

2.  begin          comment the TWOSUM problem;
                   integer I;
                   real TNEW, TOLD, TWOSUM;
                   read (TOLD);
                   I := 2;
            BOX3:  read (TNEW);
                   TWOSUM := TNEW + TOLD;
                   write ("TWOSUM =", TWOSUM);
                   TOLD := TNEW;
                   I := I + 1;
                   if  I ≤ 100  then go to BOX3;
    end
```

```
3.  begin           comment the altsum problem;
                    integer I;
                    real TOLDER, TOLD, TNEW, ALTSUM;
                    read (TOLDER);
                    read (TOLD);
                    I := 3;
             BOX4:  read (TNEW);
                    ALTSUM := TOLDER + TNEW;
                    write ("ALTSUM =", ALTSUM);
                    TOLDER := TOLD;
                    TOLD := TNEW;
                    I := I + 1;
                    if  I ≤ 100  then go to BOX4;
    end

4.  begin           comment moving average;
                    real TOLDER, TOLD, TNEW, AVERAGE;
                    integer I, k;
                    read (k);
                    read (TOLDER);
                    read (TOLD);
                    I := 3;
             BOX5:  read (TNEW);
                    if I ≥ k  then
                        begin if TNEW < TOLD then
                            begin AVERAGE := (TOLD + TOLDER)/2;
                                go to BOX11;
                            end
                                else AVERAGE := (TOLDER + TOLD + TNEW)/3;
             BOX11:                 write (AVERAGE);
                            if I ≥ 100  then go to HALT;
                        end;
                    TOLDER := TOLD;
                    TOLD := TNEW;
                    I := I + 1;
                    go to BOX5;
                    HALT:
    end
```

```
5.  begin          comment the regions table;
                   integer  N, A, B, C, SUMA, SUMB;
                   write ("N", "A", "B", "C");
                   N := 0;
                   A := 1;
                   SUMA := 1;
                   B := 1;
                   SUMB := 1;
                   C := 1;
             BOX3: write (N, A, B, C);
                   if  N ≤ 15  then
                       begin  C := SUMB + 1;
                              B := SUMA + 1;
                              A := A + 1;
                              N := N + 1;
                              SUMB := SUMB + B;
                              SUMA := SUMA + A;
                              go to BOX3;
                       end;
    end
```

Answer to Exercise 3-2 Set B

```
    begin  comment least common multiple of 2 non-negative integers;
           integer C, D, A, B, r;
           read (C, D);
           A := C;
           B := D;
           if A ≤ B
                then begin
           BOX5:       if A = 0
                              then go to BOX7
                              else begin r := B - entier(B/A) × A;
                                 BOX11: B := A;
                                        A := r;
                                        go to BOX5;
                                   end;
                       end
                else begin  r := B;

                              go to BOX11
                       end;

           BOX7: if B = 0 then x := 0 else x := C × D/B;
                 write (C, D, X);
    end
```

Answers to Exercises A3-2 Set C

```
1.  begin          real x1, y1, x2, y2, length;
            BOX1:  read (x1, y1, x2, y2);
                   write (x1, y1, x2, y2);
                   length := sqrt ((x2-x1)↑ 2 + (y2-y1)↑ 2);
                   write ("The length of PQ is", length);
                   go to BOX1;
    end
```

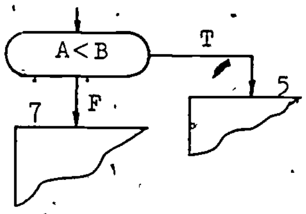
```
2.  begin           real  x1, y1, x2, y2, s;
           BOX1:  read (x1, y1, x2, y2);
                  write (x1, y1, x2, y2);
                  if  x2 = x1  then  write ("PQ  is parallel to the  y-axis")
                       else begin  s := (y2 - y1)/(x2 - x1);
                            write ("The slope of  PQ  is",  s); end;
                  go to BOX1;
    end

3.  begin           real x1, y1, x2, y2, s, delx, dely;
                  read (x1, y1, x2, y2);
                  write (x1, y1, x2, y2);
           BOX1:  read (delx);
                  write (delx);
                  if  x2 = x1
                       then begin if  delx = 0  then
                            write ("Any real number will do.")
                            else write ("No such value exists."); end
                       else begin  s := (y2 - y1)/(x2 - x1);
                                  dely := s × delx;
                                  write ("dely =" , dely); end;
                  go to BOX1;
    end

4.  begin           real x1, y1, x2, y2, dely, delx, s;
           BOX1:  read (x1, y1, x2, y2);
                  write (x1, y1, x2, y2);
                  read (dely);
                  write (dely);
                  if  y1 = y2
                  then begin if  dely = 0
                       then write ("Any real number will do.")
                       else write ("No such value exists."); end
                  else begin if  x1 = x2
                       then  delx := 0
                       else begin  s := (y2-y1)/(x2-x1);
                                   delx := dely/s; end;
                       write ("delx =", delx); end;
                       go to BOX1;
    end
```

```
5.  begin          real  x1, y1, x2, y2, s, x, y;
           BOX1:   read (x1, y1, x2, y2);
                   write (x1, y1, x2, y2);
                   read (x);
                   write (x);
                   if  x1 = x2
                   then begin if  x = x1
                        then write ("Any real number will do.")
                        else write ("No such value exists."); end
                   else begin  s := (y2-y1)/(x2-x1);
                               y := y1 + s x/(x-x1);
                               write ("y = ", y); end;
                   go to BOX1;
    end

6.  begin          real x1, y1, x2, y2, x, y, s;
           BOX1:   read (x1, y1, x2, y2);
                   write (x1, y1, x2, y2);
                   read (y);
                   write (y);
                   if  y1 = y2
                   then begin if  y = y1
                        then write ("Any real number will do.")
                        else write ("No such value exists."); end
                   else begin if  x1 = x2
                        then  x := x1
                        else begin  s := (y2-y1)/(x2-x1);
                                    x := x1 + (y-y1)/s; end;
                        write ("x = ", x); end;
                   go to BOX1;
    end
```

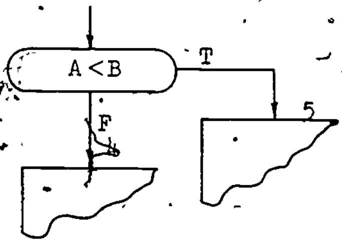
```
7.  begin       real  x1; y1, x2, y2, xint, yint, s;
          BOX1: read (x1, y1, x2, y2);
                write (x1, y1, x2, y2);
                if  x1 = x2
                then begin  write ("x-intercept is", x1);
                            write ("PQ  does not intersect y-axis."); end
                else begin if  y1 = y2
                     then begin  write ("PQ  does not intersect x-axis.");
                                 write ("y-intercept is", y1); end
                     else begin  s := (y2-y1)/(x2-x1);
                                 xint := x1 - y1/s;
                                 yint := -s × xint;
                                 write ("x-intercept is", xint);
                                 write ("y-intercept is", yint); end;
                     end;
                go to BOX1;
    end

8.  begin       real x1, y1, x2, y2, s, xint, yint;
          BOX1: read (x1, y1, x2, y2);
                write (x1, y1, x2, y2);
                if  x1 = x2
                then begin xint : = x1;
                           go to BOX 6 end
                else begin s := (y2-y1)/(x2-x1);
                           xint := x1 - y1/s;
                           yint := -x × xint
                     BOX6: if  y1 × y2 < 0
                           then write ("x-intercept is", xint)
                           else write ("PQ does not intersect the x-axis");
                           if  x1 × x2 < 0
                           then write ("y-intercept is", yint)
                           else
                           write ("PQ does not intersect the y-axis");
                end;
                go to BOX1;
    end
```

Answers to Exercises A3-3 Set A

1. Legal.

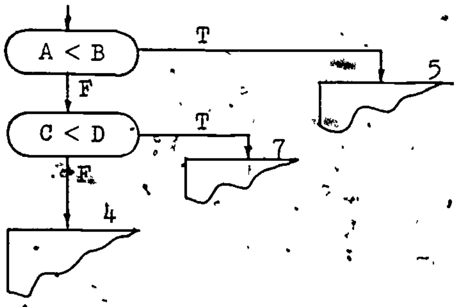


2. Legal.

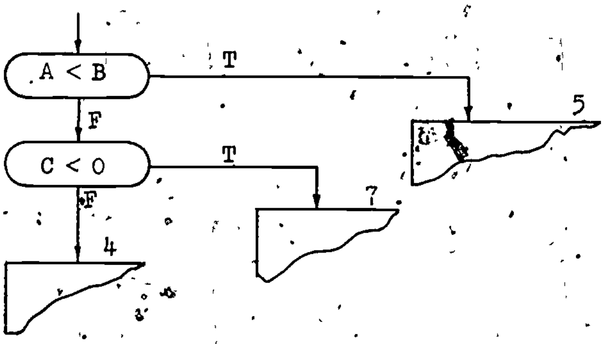


3. Illegal. Second if should be part of a compound statement like begin if C < D ...

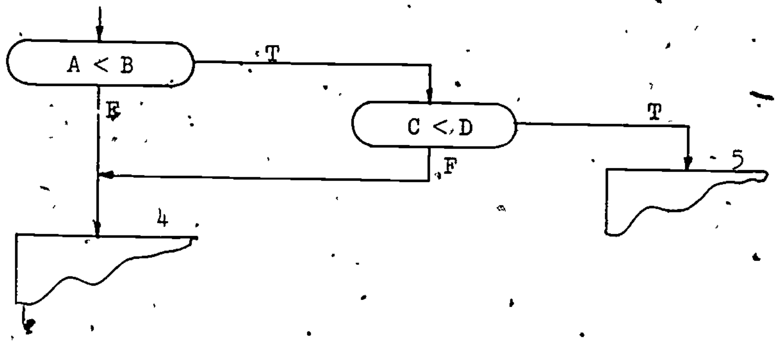
4. Legal.



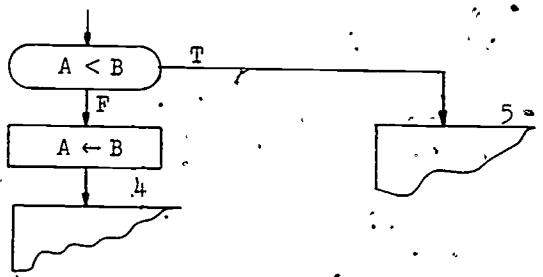
5. Legal.



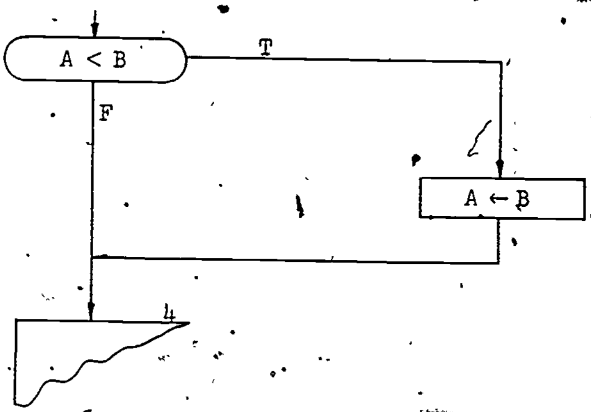
6. Legal.



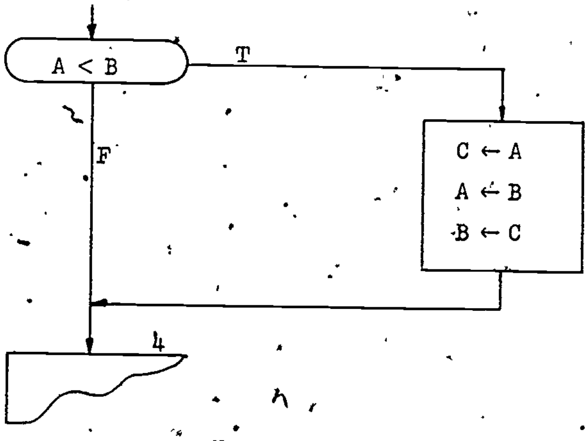
7. Legal.



8. Legal.

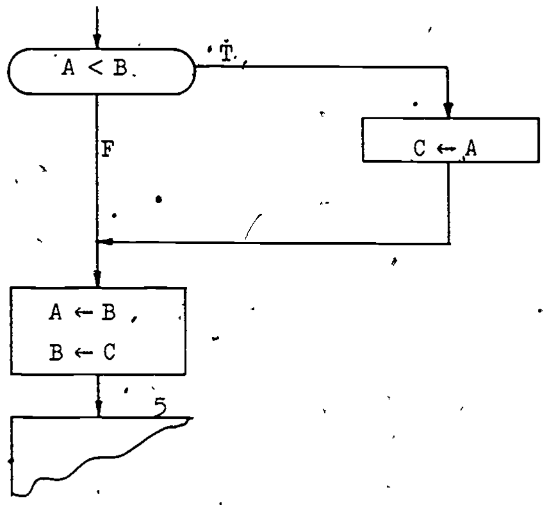


9. Legal.



10. Illegal. Need begin and end around the three assignment statements or else we should eliminate [else go to BOX5;]

11. Legal.



Answers to Exercises A3-3 Set B

1. Following the flow chart

(a)
```
if 2 ≤ x then begin
      if x ≤ 7 then go to BOX20
      else go to BOX30; end
      else go to BOX30;
```

(b) By changing the sense of the tests we have a somewhat simpler answer:
```
if x < 2 then go to BOX30;
if x > 7 then go to BOX30
     else go to BOX20;
```

2.
```
if 7 < Q then go to BOX20
else if 7 < R then go to BOX20
else if 7 < S then go to BOX20
else go to BOX30;
```

3. (a)

```
        if 1.7 < x  then go to BOX2
                    else go to BOX30;
BOX2:   if x < 8.4  then go to BOX3
                    else go to BOX30;
BOX3:   if -3.9 < y then go to BOX4
                    else go to BOX30;
BOX4:   if y < 5.4  then go to BOX20
                    else go to BOX30;
```

(b) By changing the sense of the tests we have a somewhat simpler form.

```
if 1.7 ≥ x  then go to BOX30
else if  x ≥ 8.4  then go to BOX30
else if  -3.9 ≥ y  then go to BOX30
else if  y ≥ 5.4  then go to BOX30
else go to BOX20;
```

4. (a)

```
        if xl > 0          then go to BOX2
                           else go to BOX30;
BOX2:   if .5 × xl < yl    then go to BOX3
                           else go to BOX30;
BOX3:   if yl < 2 × xl     then go to BOX20
                           else go to BOX30;
```

(b)

```
        if xl < 0          then go to BOX5
                           else go to BOX30;
BOX5:   if 2 × xl < yl     then go to BOX6
                           else go to BOX30;
BOX6:   if yl < .5 × xl    then go to BOX20
                           else go to BOX30;
```

(c)
```
            if  x1 = 0  then go to BOX30
            else if  x1 < 0  then go to BOX5
            else begin
                 if  .5 × x1 < y1  then go to BOX3
                                   else go to BOX30;
    BOX3:        if  y1 < 2 × x1   then go to BOX20
                                   else go to BOX30; end;
    BOX5:        if  2 × x1 < y1   then go to BOX6
                                   else go to BOX30;
    BOX6:        if  y1 < .5 × x1  then go to BOX20
                                   else go to BOX30;
```

Note that the above answer can be shortened by reversing the sense of the tests.

5. (a)
```
            if  x1 > 0  then go to BOX2
                        else go to BOX30;
    BOX2:   if  y1 > 0  then go to BOX3
                        else go to BOX30;
    BOX3:   if  y1 < -2/3 × x1 + 2  then go to BOX20
                                    else go to BOX30;
```

(b) an alternative solution:
```
    if  x ≤ 0  then go to BOX30;
    if  y1 ≤ 0  then go to BOX30;
    if  y1 < -2/3 × x1 + 2  then go to BOX20
                            else go to BOX30;
```

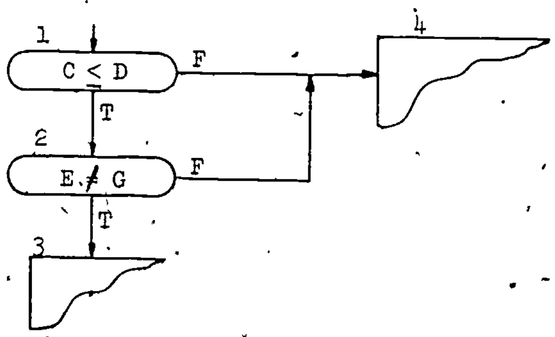
6. By reversing the sense of the tests,
```
    if  x1 < 0  then go to BOX30;
    if  .5 × (3.14159 - x1) > y1  then go to BOX30;
    if  y1 > sin(x1)  then go to BOX30
                      else go to BOX20;
```

7\.
```
if  y1 > 0  then
    begin if  y1 ≥ -4 × x1 + 16  then
          begin if  y1 ≥ 4 × x1-12  then
                go to BOX20;
          end;
    end
       else go to BOX30;
```

Answers to Exercises A3-3 Set C

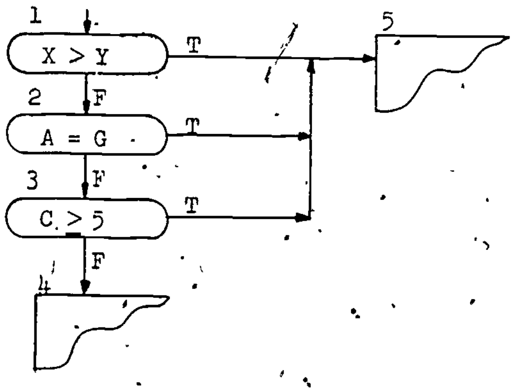
The ALGOL statements are usually easier to write from the form which is a series of F condition boxes.

1.



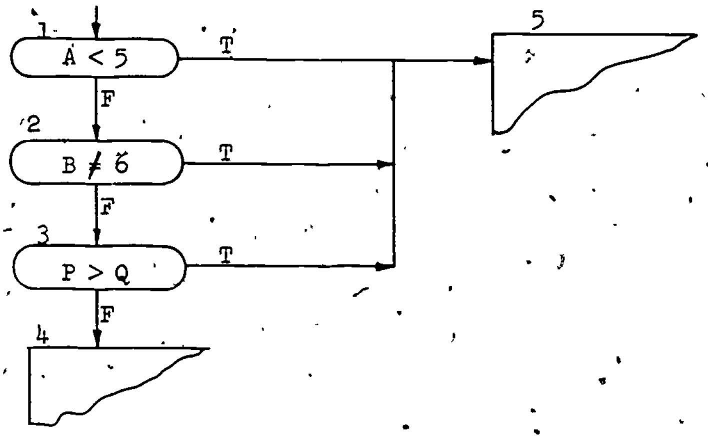
if $C > D$ then go to BOX4 else if $E = G$ then go to BOX4
else go to BOX3;

2.



if $X > Y$ then go to BOX5 else if $A = G$ then go to BOX5
else if $C \geq 5$ then go to BOX5 else go to BOX4;

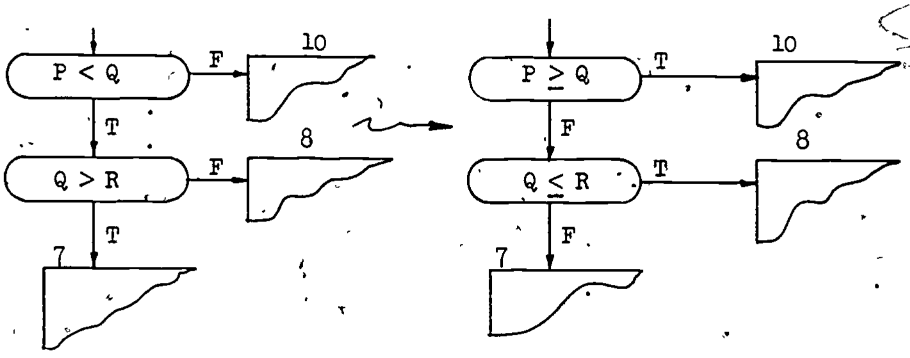
3.



<u>if</u> $A < 5$ <u>then</u> <u>go</u> <u>to</u> BOX5 <u>else</u> <u>if</u> $B \neq 6$ <u>then</u> <u>go</u> <u>to</u> BOX5 <u>else</u> <u>if</u> $P > Q$ <u>then</u> <u>go</u> <u>to</u> BOX5 <u>else</u> <u>go</u> <u>to</u> BOX4;

<u>Comment</u>: In answering Exercises 4 and 5 draw a flow chart corresponding to the given ALGOL statement. Then, if warranted, reverse the sense of one or more of the tests to make it easier to use the <u>else</u> <u>if</u> form.

4.



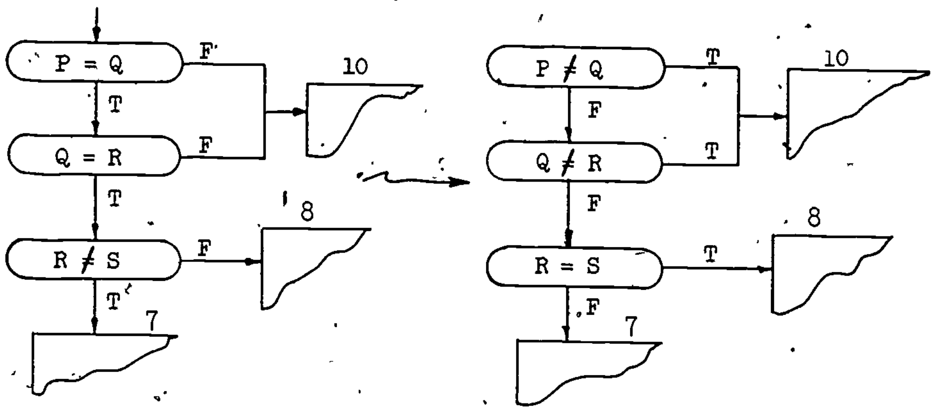
form (a)

form (b)
Reverse sense of tests

Following form (b) we have:

<u>if</u> $P \geq Q$ <u>then</u> <u>go</u> <u>to</u> BOX10 <u>else</u> <u>if</u> $Q \leq R$ <u>then</u> <u>go</u> <u>to</u> BOX8 <u>else</u> <u>go</u> <u>to</u> BOX7;

5.



form (a) form (b)

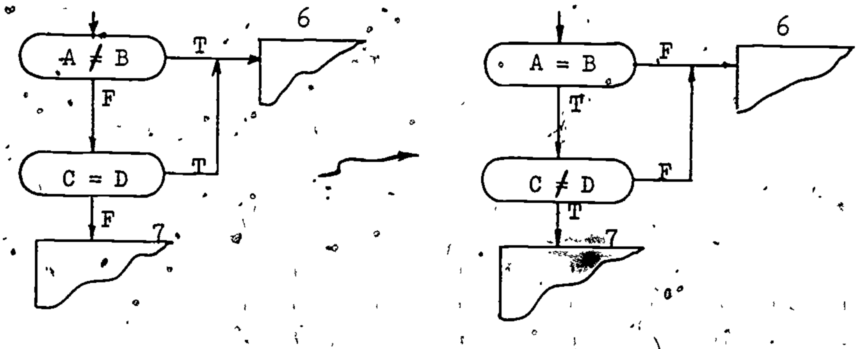
Following form (b) we have:

```
if P ≠ Q then go to BOX10 else if Q ≠ R then go to BOX10
    else if R = S then go to BOX8 else go to BOX7;
```

(Next statement is assumed to be labeled BOX10.)

Comment: In answering Exercises 6 and 7 of this set the process can be analogous to that used in answering Exercises 4 and 5. However, an alternative coding is even simpler and does not require re-flow charting as shown below.

6.



form (a) form (b)

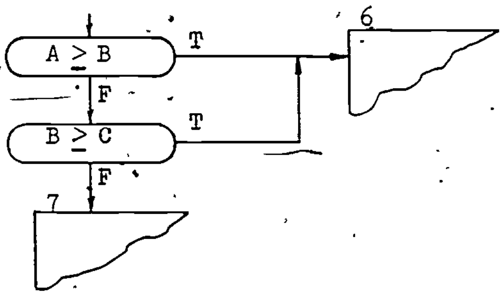
Following form(b) we have:

```
if A = B then begin if C ≠ D then go to BOX7
                                 else go to BOX6; end
                                 else go to BOX6;
```

Alternatively, following form(a), we have in two separate statements:

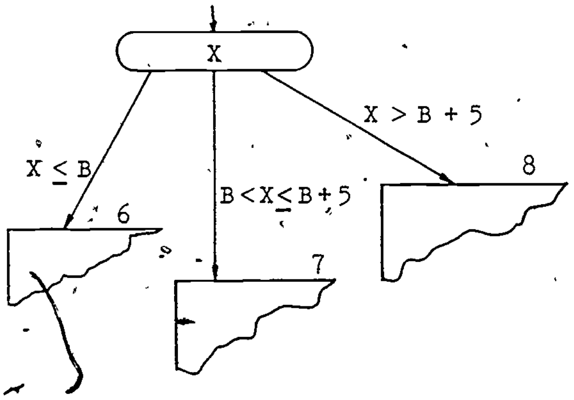
```
if A ≠ B then go to BOX6;
if C = D then go to BOX6 else go to BOX7;
```

7.

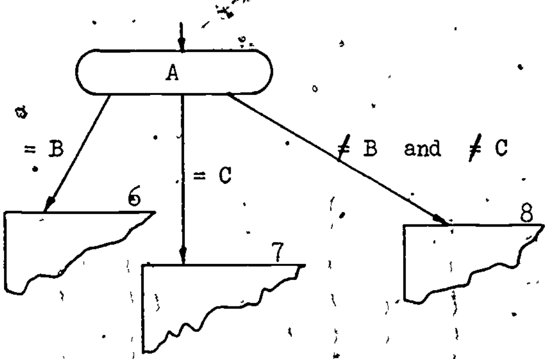


<u>if</u> A ≥ B <u>then</u> <u>go</u> <u>to</u> BOX6;
<u>if</u> B ≥ C <u>then</u> <u>go</u> <u>to</u> BOX6 <u>else</u> <u>go</u> <u>to</u> BOX7;

8.



9.



10. <u>if</u> z < 5 <u>then</u> <u>go</u> <u>to</u> BOX2 <u>else</u> <u>if</u> z ≤ 10
<u>then</u> <u>go</u> <u>to</u> BOX3 <u>else</u> <u>go</u> <u>to</u> BOX4;

Answers to Exercises A3-3, Set D

```
1.  begin        real xl, yl;
          BOX1:  read (xl, yl);
                 write (xl, yl);
                 if xl = 0 then
                    begin if yl = 0
                    then write ("P is the origin.")
                    else write ("P lies on the y-axis"); end
                 else if xl < 0 then
                    begin if yl = 0 then go to BOX6
                          else if yl < 0 then write ("3")
                          else write ("2"); end
                 else
                    begin if yl = 0
                          then BOX6: write ("P lies on the x-axis")
                          else if yl < 0
                          then write ("4")
                          else write ("1"); end;
                 go to BOX1;
      end

2.  begin        integer S, m, p, k;
          BOX1:  read (S, m);
                 k := m + S - entier((m + S)/4) × 4;
                 if k = 0 then p := -20
                 else if k = 1 then p := -30
                 else if k = 2 then p := 0
                 else p := 50;
                 write (p);
                 go to BOX1;
      end
```

Answers to Exercises A3-5, Set A

1. X[5]
2. Z[N]
3. CHAR[I]
4. B[i + 2]

Answers to Exercises A3-5 Set B

1.
```
              real array  A[1:50];
              integer  i;
              i := 1;
       BOXA:  read (A[i]);
              i := i + 1;
              if  i ≤ k  then go to BOXA;
```

Note k is a constant not a variable for the purpose of this problem because it is given a value before we start. Therefore k is not included in the integer declaration list.

2.
```
              real array  B[1:125];
              integer  j;
              j := 5;
       BOXB:  read (B[j]);
              j := j + 2;
              if  j ≤ n  then go to BOXB;
```

3.
```
              real array  A[1:50], B[1:50];
              integer i;
              i := 10;
       BOXA:  read (A[i]);
              i := i + 1;
              if  i ≤ n  then go to BOXA;
              i := 10;
       BOXB:  read (B[i]);
              i := i + 2;
              if  i ≤ n  then go to BOXB;
```

Answers to Exercises A3-5 Set C

```
1.  begin          comment  carnival wheel with subscripts;
                   integer array  P[1:4];
                   integer  s, m, k;
           BOX1:   read (s, m);
                   k ← m + s - entier((m + s)/4) × 4;
                   write (p[k + 1]);
    end

2.  begin          comment  Figure 3-25;
                   integer  i, any, k;
                   real array  b[1:100];
                   real c;
                   i := 1;
                   any := 0;
                   read (c);
                   k := 1;
          AGAIN:   read (b[k]);
                   k := k + 1;
                   if  k ≤ 100  then go to AGAIN;
           BOX4:   if  b[i] ≥ c
                       then begin any := 1; write (i, b[i]); end;
                   i ← i + 1;
                   if  i ≤ 100  then go to BOX4;
                   if  any = 0  then write ("NONE");
    end
```

Answer to Exercise A3-5 Set C

```
3.  begin        comment finding the actual degree of a polynomial;
                 integer  n, i;
                 real array A[0:50];
            BOX1: read (n);
                 i := 0;
            BOX2: read (A[i]);
                 i := i + 1;
                 if  i ≤ n  then go to BOX2;
            BOX3: if  A[n] = 0
                    then begin  n := n - 1;
                         if  n ≥ 0  then go to BOX3
                                    else begin write (n);
                                               go to FIN;
                                         end;
                         end
                    else begin write (n);
                               i := 0;
            BOX6:              write (A[i]);
                               i := i + 1;
                               if  i < n  then go to BOX6;
                         end;
             FIN: go to BOX1;
    end
```

Answer to Exercise A3-5 Set D

```
begin          comment orchestraville;
               real array A[1:125];
               integer n, k, copy, mid, mad;
               real meed;
      BOX1:    read (n);
               k := 1;
      BOX2:    read A[k];
               k := k + 1;
               if k ≤ n then go to BOX2;
      BOX3:    k := 1;
      BOX4:    if A[k] > A[k + 1]
                  then begin copy := A[k];
                             A[k] := A[k + 1];
                             A[k + 1] := copy;
                             go to BOX3;
                       end;
               k := k + 1;
               if k < n then go to BOX4;
               mid := entier(n/2);
               if mid × 2 = n then mad := mid + 1
                              else mad := mid;
               meed := .5 × (A[mid] + A[mad]);
               write ("MEDIAN AGE IS", meed, "YOUNGEST IS", A[1],
                    "OLDEST IS", A[n]);
               go to BOX1;
end
```

Answers to Exercises A3-6

1.
```
             real array  P[1:22, 1:27];
             real  COLSUM;
             integer  I, K;
             COLSUM := 0;
             I := 1;
      BOX3:  if  I = 12  then go to BOX5;
             COLSUM := COLSUM + P[I, K];
      BOX5:  if  I < 22  then go to BOX6
                         else write COLSUM;
             go to BOX8;
      BOX6:  I := I + 1;
             go to BOX3;
```

2.
```
             real array  P[1:22, 1:27];
             integer  J, L, M;
             J := 1;
      BOX2:  P[L,J] := P[L,J] + P[M,J];
             if  J < 27  then begin
                         J := J + 1;
                         go to BOX2; end;
```

3.
```
             real array. P[1:22, 1:27];
             integer  L, J, K, M;
             J := 1
      BOX2:  if  J = K  then go to BOX4;
             P[L,J] := P[L,J] + 2 × P[M,J];
      BOX4:  if  J < 27  then begin
                         J := J+1;
                         go to BOX2; end;
```

4.

```
         real array P[1:22,1:27];
         integer I, J, L, M;
         real COPY;
         I := 1;
BOX2:    COPY := P[L,J];
         P[L,J] := P[M,J];
         P[M,J] := COPY;
         if J < 27 then
            begin J := J + 1;
                  go to BOX2;
            end;
```

5.

```
         real array P[1:22, 1:27];
         integer J;
         real MAX;
         J := 1;
         MAX := 0;
BOX3:    if abs(P[L,J]) > abs(MAX) then MAX : = P[L,J];
         if J < 27 then begin J := J + 1;
                              go to BOX3;
                        end;
         J := 1;
BOX8:    P[L,J] := P[L,J]/MAX;
         if J < 27 then begin J := J + 1;
                              go to BOX8;
                        end;
```

Chapter TA4

LOOPING

Summary of Chapter A4

The main sections of this chapter are:

A4-1	The "for clause" and the "for statement"
A4-2	Illustrative examples
A4-3	Table-look-up
A4-4	Nested loops

This chapter follows closely Chapter 4 of the flow chart text.

A4-1 The "for clause".

The "for clause" in ALGOL is introduced and shown to be the equivalent of the iteration box and the for statement equivalent to the whole loop, that is, governed by the iteration box and including the iteration box. The portion of the for statement which follows the for clause is shown to be any ALGOL statement that the student has already learned about thus far, i.e., assignment, input, output, or if statement. Moreover, it may also be a compound statement. The latter possibility makes it possible for any loop, which begins with an iteration box, to be described with a single for statement.

In the exercises to this section the student is shown how to use a for statement to code the input or output of vectors, where the flow chart notation is something like:

$$\{P_i, i = 1(1)4\}$$

A4-2 Illustrative Examples

The examples of the iteration box used for simple loops in Section 4-2 are mirrored in this section using for statements.

A4-3 Table-look-up

The main purpose of this section is to provide the student additional opportunity to see how complicated algorithms, which involve loops and iteration boxes, are converted or "transliterated" from flow chart to ALGOL.

A4-4 Nested loops

Details of nesting for statements are described to mirror the nesting of iteration loops in the flow charts. We show the student how the statement that follows the for clause can be another for statement or contain one (if it is a compound).

We also show the student how to express the input or output of an entire matrix by nested for statements (Figure-A4-11).

Additional coding practice is gained in the exercises which call for translation of flow charts to ALGOL.

Answers to Exercises A4-1

```
1.  begin    integer N, I, ID;
             real A, B, C, D;
     BOXO:   read(N);
             for I := 1 step 1 until N do
                  begin    read (ID, A, B, C);
                           D := sqrt(A↑2 + B↑2 + C↑2);
                           write (ID, A, B, C, D); end;
                  write ("END□OF□TABLE");
                  go to BOXO;
    end

2.  begin    integer LTERM, NLT, COPY, I, S;
             LTERM := 2;
             NLT := 1;
             S := 1;
             for I := 1 step 1 until 60 do
                begin
                  write (I, LTERM, S);
                  S := S+ NLT;
                  COPY := LTERM;
                  LTERM := LTERM + NLT;
                  NLT := COPY;
                end;
    end

3.  begin    integer array P[1:4];
             integer N, SUM, L, s, m, k;
             for I := 1 step 1 until 4 do read (P[I]);
             read (N);
             SUM := 0;
             for L := 1 step 1 until N do
                begin
                  read (s, m);
                  k := m + s - entier((m+s)/4)×4;
                  SUM := SUM + P[k + 1];
                end;
             write ("after", N, "spins, your net winnings are", SUM, "points");
    end
```

```
4.  begin   integer i, N;
            real PAYROLL, WAGES;
            real array T[1:100], R[1:100];
            read (N);
            PAYROLL := 0;
            for i := 1 step 1 until N do read (T[i]);
            for i := 1 step 1 until N do read (R[i]);
            for i := 1 step 1 until N do
                begin WAGES := R[i] × T[i];
                      PAYROLL := PAYROLL + WAGES;
                      write (i, WAGES);
                end;
            write(PAYROLL);
    end
```

Answers to Exercises A4-2 Set A

1. The first statement should read

```
MAX := abs(A[1]);
```

The for clause is improperly written. It should read

```
for J := 2 step 1 until N do
```

Otherwise, O.K.

2. There are two errors:

The first statement lacks a semicolon.

The if statement has a semicolon before then--it must be removed! The presence of this unwanted semicolon makes an ill-formed if statement!

3. There are two errors:

The for clause has an unwanted semicolon after do.

There are two statements which should be repeated under control of the counter k. We need to make a compound statement:

```
begin  FACT := K × FACT;
       write (K, FACT);
end
```

4. There is one error:

The write statement must be "stuffed" inside the compound, i.e., ahead of the last end

The first two statements are written as a compound statement. While this is quite unnecessary, it is not illegal and cannot be considered an error as it does not change the sense of this code.

Answers to Exercises A4-2 Set B

```
1.  for I := 1 step 1 until N do
       begin COPY := P[I];
             P[I] := Q[I];
             Q[I] := COPY;
       end;
```

```
2.   for I := 2 step 2 until N do
        begin COPY := P[I];
              P[I] := Q[I];
              Q[I] := COPY;
        end;

3.   for I := 5 step 3 until N do
        begin COPY := P[I];
              P[I] := Q[I];
              Q[I] := COPY;
        end;

4.   ND2 := N/2;
     for I : = 1 step 1 until NO2 do
        Q[I] := P[I];

     alternatively,
     for I := 1 step 1 until entier(N/2) do
        Q[I] := P[I];

5.   NO2 := N/2 ;
     for I := 1 step 1 until NO2 do
        Q[I] := P[NO2 + I];

     alternatively,
     for I := 1 step 1 until N/2 do
        Q[I] := P[N/2 + I];

6.   NO2B : = entier (N/2);
     if NO2B = N/2 then K : = NO2B
                  else K := NO2B + 1;
     for I := 1 step 1 until NO2B do
        Q[I] := P[K + 1];

7.   for I := N step -1 until N - K + 1 do
        P[I + 2] := P[I];

8(a).   SUMCUB := 0;
        for I := 1 step 1 until 100 do
           SUMCUB := SUMCUB + P[I]↑3;
```

```
8(b).  SUMNEG := 0;
       for I := 1 step 1 until 100 do
       if P[I] < 0 then SUMNEG := SUMNEG + P[I];

8(c).  SUMCUB := 0;
       SUMNEG := 0;
       SUMBIG := 0;
       for I := 1 step 1 until 100 do
          begin
              SUMCUB := SUMCUB + P[I]↑3;
              if P[I] < 0 then SUMNEG := SUMNEG + P[I];
              if abs(P[I]) > 50 then SUMBIG := SUMBIG + abs P[I];
          end;

9.   COLSUM := 0;
     for I := 1 step 1 until 22 do
      if I ≠ 12 then COLSUM := COLSUM + P[I, K];
     write (COLSUM);

10.  for J := 1 step 1 until 27 do
       P[L, J] := P[L, J] + P[M, J];

11.  for J := 1 step 1 until 27 do
        if J ≠ K then P[L, J] := P[L, J] + 2 × P[M, J];

12.          for I := 1 step 1 until N do
             if abs(P[I]) ≥ 50 then go to BOX3;
             ANY := 0;
             go to BOX5;
     BOX3:   W := P[I];
             ANY := 1;
     BOX5:   ~~~~~
```

Comment on Problem 12--suggested code:

The ALGOL code proposed in the student text for this exercise is wrong because Box 3 of the flow chart, which is outside the loop, has been "gathered" into the loop in the proposed ALGOL code. In the flow chart once you enter Box 3 you have left the for-statement.

13.

```
          W := 50;
          for I := N step -1 until 1 do
          if abs(P[I]) > 50 then go to BOX4;
          go to BOX5;
   BOX4:  W := P[I];
   BOX5:  ~~~~~~
```

or alternatively,

```
          W := 50;
          for I := 1 step 1 until N do
          if abs(P[N - I + 1]) > 50 then go to BOX4;
          go to BOX5;
   BOX4:  W := P[I];
   BOX5:  ~~~~~~
```

14.

```
                T := 0;
Note: all      {for I := 1 step 1 until N do
one for        {if abs(P[I]) < abs(M) then begin
statement      {if abs (P[I]) > abs(T) then T := P[I]; end;
                if T = 0 then begin write ("NONE"); go to HALT; end;
   BOX8:        ~~~~~~
```

It's assumed that the empty statement labeled HALT is to be found at the end of the program. A similar example was used in the student text at the beginning of Section A3-2.

15.

```
                  for I := 1 step 1 until N do
                     if P[I] < M then go to BOX3;
                  write ("NONE");
                  go to HALT;
           BOX3:  T := P[I];
           BOX4:  for K := I + 1 step 1 until N do
Note: all  {      if T < P[K] then begin
one for    {      if P[K] < M then T := P[K]; end;
statement  {
           BOX8:  ~~~~~~
```

Again it's assumed that an empty statement labeled HALT is to be found at the end of the program.

16.
```
        SMALL := Q[L, 1];
        for J := 2 step 1 until N do
        if SMALL > Q[L, J] then SMALL := Q[L, J];
```

17.
```
        for I := M step -1 until 1 do
        if Q[I, R] ≥ T then go to BOX3;
BOX4:   ROW := 0;
        go to BOX5;
BOX3:   ROW := I;
        BIG := Q[I, R];
BOX5:   ~~~
```

Answers to Exercises A4-2 Set C

1.
```
begin    real array X[1:50];
         real A, NUM;
         integer I, J, N;
         read (N);
         for I := 1 step 1 until N do read (X[I]);
         read (A);
         NUM := X[1] - A;
         for J := 2 step 1 until N do
           NUM := NUM × (X[J] - A);
         write (NUM);
end
```

1b.
```
begin    real array X[1:50];
         real A, NUM;
         integer K, I, J, N;
         read (N);
         for I := 1 step 1 until N do read (X[I]);
         read (K, A);
         NUM := 1;
         for J := 1 step 1 until N do
           if J ≠ K then
              NUM := NUM × (X[J] - A);
         write (NUM);
end
```

```
2.  begin    real array X[1:50];
             real DEN;
             integer K, I, J, N;
             read (N);
             for I := 1 step 1 until N do read (X[I]);
             read (K);
             DEN := 1;
             for J := 1 step 1 until N do
                if J ≠ K then
                  DEN := DEN × (X[J] - X[K]);
             write (DEN);
    end

3b. begin    integer array P[1:4];
             integer CV, SUM, L, s, m, k;
             for i := 1 step 1 until 4 do read (P[I]);
             read (CV);
             SUM := 0;
             for L := 1 step 1 until 1000 do
                begin
                    read (s, m);
                    k := m + s - entier((m+s)/4)×4;
                    SUM := SUM + P[k + 1];
                    if abs(SUM) > CV then begin
                      write (L, CV, SUM);
                      go to HALT; end;
                end;
             write ("ERROR");
       HALT:;
    end
```

Answer to Exercise A4-3

```
begin  comment Look up in an unsorted Table;
       real array X[1:200], Y[1:200];
       real A;
       integer K, N, LO, HI, I;
       read (N);
       for I := 1 step 1 until N do read (X[K], Y[K]);
       LO := 0;
       HI := N + 1;
       read (X[LO], X[HI]);
       read (A);
       if X[LO] ≤ A then begin
          if A ≤ X[HI] then go to BOX7; end;
       write (A, "is not in the range of the Table");
       go to HALT;
BOX7:  for I := 1 step 1 until N do
       begin if X[I] ≤ A then
          begin if X[I] ≥ X[LO] then LO := I; end
             else if X[I] ≤ X[HI] then HI := I;
       end;
       write (X[LO], Y[LO], A, X[HI], Y[HI]);
  HALT: ;
end
```

Answers to Exercises A4-4 Set A

1.

```
BIG := 0;
for I := 1 step 1 until M do
  for J := 1 step 1 until N do
    if abs(BIG) < abs(P[I, J]) then
       BIG := P[I, J];
write (BIG);
```

2.

```
LARGE := P[1, 1];
ROW := 1;
COL := 1;
for I := 1 step 1 until M do                    one great big
  for J := 1 step 1 until N do                  for statement
    if LARGE < P[I, J] then
       begin  LARGE := P[I, J];
              ROW := I;
              COL := J;
       end;
write (LARGE, ROW, COL);
```

3.

```
                LEAST := P[1, 1];
                ZTALY := 0;
                for I := 1 step 2 until M do
one big for       for J := 2 step 2 until N do
statement           if P[I, J] = 0 then ZTALY := ZTALY + 1
                    else if LEAST > P[I, J] then LEAST := P[I, J];
                write (LEAST, ZTALY);
```

4.

```
for I := 2 step 1 until M do
  for J := 1 step 1 until N do
    P[I, J] := P[I, J] + T × P[1, J];
```

5.

```
for J := 1 step 1 until N do
begin
  MIN := P[1, J];
  ROW := 1;
  for I := 2 step 1 until M do
    if MIN ≥ P[I, J] then
      begin
        MIN := P[I, J];
        ROW := I;
      end;
  write (MIN, J, ROW);
end;
```

one great big for statement

6.

```
SUM1 := 0;
for I := 2 step 1 until M do
  for J := 1 step 1 until I - 1 do
    SUM1 := SUM1 + P[I, J];
```

7.

```
SUM2 := 0;
for I := 1 step 1 until M - 1 do
  for J := 1 step 1 until M do
    SUM2 := SUM2 + P[I, J];
```

8.

```
       ANY := 0;
       for J := M step -1 until 3 do
         begin
           LAST := P[1, J];
           for I := 2 step 1 until J - 1 do
             if abs(P[I,J]) ≥ 2 × LAST then go to BOX7
             else LAST := P[I, J];
BOX7:      write (P[I, J], I, J);
           ANY := 1;
         end;
       if ANY = 0 then write ("NONE");
```

Answers to Exercises A4-4 Set B

```
1(a).   begin   comment Number of non-congruent triangles whose
                sides are of integer length less than 100;
                Integer I, J; S;
                S := 0;
                for I := 1 step 1 until 100 do
                   for J := 1 + entier (I/2) step 1 until I do
                      S := S + 2 × J - I;
                write(S);
        end

 (b).   begin   comment Accumulated perimeters for the  S  triangles
                counted by preceding program;
                Integer I, J, K, P;
                P := 0;
                for I := 1 step 1 until 100 do
                   for  J := 1 + entier(I/2) step 1 until I do
                      for K := I - J + 1 step 1 until J do
                         P := P + I + J + K;
                write(P);
        end
```

Answers to Exercises A4-4 Set C

1.

```
        begin comment prime Factorization Algorithm;
              integer N, K;
              read (N);
              for K := 2 step 1 until sqrt(N) do
                BOX3: if N = entier(N/K) × K then
                      begin write (K);
                            N := N/K;
                            go to BOX3;
                      end;
              if N ≠ 1 then write (N);
        end
```

one for statement

tricky way to write a one statement loop without use of a for statement

2.

```
        begin comment shuttle interchange sorting;
              real array A[1:500];
              real COPY;
              integer I, J, K, N;
              read (N);
              for I := 1 step 1 until N do read (A[I]);
              for J := 1 step 1 until N - 1 do
                begin
                  if A[J] ≤ A[J + 1] then go to JUNCT;
                  COPY := A[J];
                  A[J] := A[J + 1];
                  A[J + 1]  := COPY;
                  for K := J - 1 step -1 until 1 do
                    if A[K] ≤ A[K + 1] then go to JUNCT
                    else begin COPY := A[K];
                               A[K] := A[K + 1];
                               A[K + 1] := COPY;
                         end;
                  JUNCT: ;
                end;
              for I := 1 step 1 until N do write (A[I]);
        end
```

one for statement

3.
```
begin  comment sleeper Fig. 4-35;
       real array A[1:500];
       real COPY;
       integer I, J, K, N;
       read (N);
       for K := 1 step 1 until N do read (A[K]);
       for I := 1 step 1 until N - 1 do
         for J := 1 step 1 until N - I do
           if A[J] > A[J + 1] then
             begin COPY := A[J];
                   A[J] := A[J + 1];
                   A[J + 1] := COPY;
             end;
       for K := 1 step 1 until N do write (A[K]);
end
```

4.
```
begin  comment longest decreasing subsequence;
       real array A[1:100];
       integer array B[1:100];
       integer N, I; MAXINC, J, K;
       read (N);
       for I := 1 step 1 until N do read (A[I]);
       MAXINC := 1;
       for J := 1 step 1 until N do
         begin B[J] := 1;
               for K := 1 step 1 until J - 1 do
                 if A[K] > A[J] then begin
                   if B[J] < B[K] + 1 then
                      B[J] := B[K] +1; end;
               if MAXINC < B[J] then
                  MAXINC := B[J];
         end;
       write (MAXINC);
end
```

Chapter TA5

PROCEDURES

Answers to Exercises A5-1

```
1.  real procedure  cubert(a); real a;
        begin
             real  g,h;
             g:=1;
    loop:    h:=(2 × g + a/g ↑ 2)/3;
             if  abs(h-g) ≥ .0001  then
                  begin  g:=h;
                         go to loop;
                  end;
             cubert := h;
        end
```

```
2.  real procedure  f(x);
        real  x;
        f:=(3 × x - 2) × x + 1;
```

```
3.  real procedure  absol(x);
        real  x;
        if  x < 0  then absol := -x  else absol := x;
```

Answers to Exercises A5-3 Set A

```
1.    (a)  real procedure  f(x,y); real x,y;
                f:=((x ↑ 3 + y) ↑ 2 + 5)/(abs(x) + 2);

      (b)  z:=f(r,s) + 6 × T;

2.    integer procedure right(a,b,c); real a,b,c;
           begin  right := 0;
                if  c ≤ 0  then go to error;
                if  b ≤ 0  then go to error;
                if  a ≤ 0  then go to error;
                if  c × c = a × a + b × b  then right:=1 else
                if  a × a = b × b + c × c  then right:=1 else
                if  b × b = a × a + c × c  then right:=1;
      error:
           end

3.    (a)  real procedure  max(x,y,z); real x, y, z;
                begin  real  lrgst;
                     lrgst:=x;
                     if  lrgst < y  then  lrgst:=y;
                     if  lrgst < z  then  lrgst:=z;
                     max:=lrgst;
                end

      (b)  begin
                real  lrgst;
                real procedure  max(x,y,z); real x, y, z;
                     comment  put procedure body here;
                read(A,B,C);
                lrgst := max(A,B,C);
                write(lrgst);
           end
```

4. Let quad = 0 indicate the error exit.

```
integer procedure quad(x,y); real x,y;
    begin real z;
        if x = 0 then z:=0;
        if x > 0 then
            begin if y = 0 then z:=0;
                  if y > 0 then z:=1;
                  if y < 0 then z:=4;
            end;
        if x < 0 then
            begin if y = 0 then z:=0;
                  if y > 0 then z:=2;
                  if y < 0 then z:=3;
            end;
        quad:=z;
    end
```

5.

```
integer procedure insect(x1,y1,r1,x2,y2,r2);
    real x1, y1, r1, x2, y2, r2;
    begin real dist;
        if r1 ≤ 0 then insect:=-1 else
        if r2 ≤ 0 then insect:=-1 else
        begin
            dist:=sqrt((x2-x1) ↑ 2 + (y2-y1) ↑ 2);
            if dist = 0 then
                begin if r1 = r2 then insect:=100 else
                         insect:=0 end else
            if dist = r1 + r2 then insect := 1 else
            if dist = abs(r1 - r2) then insect := 1 else
            if dist < abs(r1 - r2) then insect := 0 else
            if dist > r1 + r2 then insect := 0 else
            insect := 2;
        end;
    end
```

```
6.  real procedure anyroot(a,n); real a; integer n;
        begin
            real h,g; integer i;
            g := 1;
            for i := 1 step 1 until 10 do
            begin
                h := a/g↑(n-1);
                if abs(g-h) < .0001 then go to fin;
                g := ((n-1) × g + h)/n;
            end;
        fin: anyroot := g;
        end

7.  (a) real procedure irate(n,R,L); real R,L; integer n;
            begin real rest, pay;
                rest := .01 × R;
                pay := L;
                for i := 1 step 1 until n do
                    pay := pay × (1 + rest);
                irate := pay;
            end
```

Alternate solution:

```
        real procedure irate (n,R,L);
            real R,L; integer n;
            irate := L × (1 + R/100)↑n;
```

Answers to Exercises A5-3 Set B

1.
```
integer procedure GCD(A,B);
    integer A,B;
    begin integer r;
        if A > B then
            begin r := B;
                  B := A;
                  A := r;
            end;
  again:  if A ≠ 0 then
            begin r := B - A × entier(B/A);
                  B := A;
                  A := r;
                  go to again;
            end;
        GCD := B;
    end
```

2.
```
integer procedure GCF(A,B,C);
    integer A,B,C;
    begin integer x;
        x := GCD(A,B);
      GCF := GCD(x,C);
    end
```

Comment: The function procedure GCD would have to be declared in the head of the main program along with the declaration of procedure GCF.

3. (a)
```
begin comment number of non-similar triangles;
        integer S,I,J,K;
        integer procedure GCD(A,B);
            comment put the rest of the declaration of
                    procedure GCD here;
        integer procedure GCF(A,B,C);
            comment put the rest of the declaration of
                    procedure GCF here;
        S := 0;
        for I := 1 step 1 until 100 do
            for J := 1 + entier(I/2) step 1 until I do
                for K := I - J + 1 step 1 until J do
        here:           if GCF(I,J,K) = 1 then  S := S + 1;
        write(S);
end
```

(b) Replace the statement labeled "here" with

```
        here:           if GCF(I,J,K) = 1 then S := S + I + J + K;
```

Comment: In case students run this problem and also problem 4 on the computer, you will probably want them to cut down the size of the problem. Otherwise, excessive computer time may be required. For example, triangles whose lengths are less than 50 might be enough. Likewise, in problem 4, you could consider all numbers less than 10^6 instead of 10^9.

```
4.  begin
          integer I,J,K,L,TEST;
          integer array CUBE[1:1000];
          comment place declaration of integer procedure GCD here;
          comment place declaration of integer procedure GCF here;
          for I := 1 step 1 until 999 do
               begin CUBE[I] := I × I × I;
                      for J := 1 step 1 until I do
                           begin TEST := CUBE[I] + CUBE[J];
                                  if TEST ≥ 10↑9 then go to over;
                                          K := I - 1;
                                          L := J + 1;
          back:                   if L > K then go to again;
                                  if CUBE[L] + CUBE[K] = TEST
                                          then begin if GCF(J,K,L) = 1
                                                  then write(I,J,TEST,K,L);
                                                          L := L + 1;
                                                          K := K - 1;
                                                          go to back;
                                                  end
                           else if CUBE[L] + CUBE[K] < TEST
                                  then begin L := L + 1;
                                                          go to back;
                                                  end
                           else if CUBE[L] + CUBE[K] > TEST
                                  then begin K := K - 1;
                                                          go to back;
                                          end;
       again:               end;
       over; end;
   end
```

TA

Answers to Exercises A5-4 Set A

```
1.  procedure absol(x,absx); real x, absx;
        begin if x < 0 then absx:=-x else absx:=x; end

2.  (a) procedure cxadd(a1,b1,a2,b2,a,b); real a1, b1, a2, b2, a, b;
            begin a:=a1 + a2;
                  b:=b1 + b2;
            end

    (b) procedure cxsub(a1,b1,a2,b2,a,b); real a1, b1, a2, b2, a, b;
            begin a:=a1 - a2;
                  b:=b1 - b2;
            end

    (c) procedure cxmult(a1,b1,a2,b2,a,b); real a1, b1, a2, b2, a, b;
            begin a:=a1 × a2 - b1 × b2;
                  b:=a1 × b2 + a2 × b1;
            end

    (d) procedure cxdiv(a1,b1,a2,b2,a,b); real a1, b1, a2, b2, a, b;
            begin real denom;
                denom := a2 × a2 + b2 × b2;
                    a:= (a1 × a2 + b1 × b2)/denom;
                    b:= (a2 × b1 - a1 × b2)/denom;
            end

    (e) begin real a1, b1, a2, b2, oper;
            comment place declaration of procedures cxadd, cxsub, cxmult,
                    and cxdiv here;
                read(a1,b1,a2,b2,oper);
                write(a1,b1,a2,b2,oper);
                if oper = 1 then cxadd(a1,b1,a2,b2,a,b) else
                if oper = 2 then cxsub(a1,b1,a2,b2,a,b) else
                if oper = 3 then cxmult(a1,b1,a2,b2,a,b) else
                if oper = 4 then cxdiv(a1,b1,a2,b2,a,b);
                write(a,"+",b,"i");
        end
```

```
3.  procedure  sort2(k,A,B,error);
      integer  k, error; array A,B;
      begin real copy; integer i;
         if  k ≤ 0  then
         begin  error:=1;
              go to return;
         end;
         error:=0;
   again:  for  i:=1 step 1 until k - 1 do
         begin if  A[i] > A[i+1] then
              begin copy:=A[i];
                   A[i]:=A[i+1];
                   A[i+1]:=copy;
                   copy:=B[i];
                   B[i]:=B[i+1];
                   B[i+1]:=copy;
                   go to again;
              end;
         end;
     return:
     end

4.   (a)  procedure count(n,countfac);
         integer  n, countfac;
         comment A negative value of countfac indicates N ≤ 0;
         begin
              real bound; integer k;
              if n ≤ 0 then
                   begin countfac := -1;
                        go to return;
                   end;
              countfac := 0;
              bound := sqrt(n);
              for k := 1 step 1 until bound -1 do
                   if n = k × entier(n/k) then
                        countfac := countfac + 2;
              if n = k × k then countfac := countfac + 1;
              return:
        end
```

4. (b)
```
begin integer n, fac;
    comment place procedure count here;
    for n:=1 step 1 until 1000 do
        begin count(n,fac);
            if fac = 2 then write (n);
        end;
end
```
5. (a)
```
procedure aliquot(number,n,parts);
    integer number, n; integer array parts;
    begin real bound; integer k;
        if number ≤ 0 then
            begin n:=-1; go to return; end;
        n:=1;
        parts[1]:=1;
        if number ≤ 3 then go to return;
        bound:=sqrt(number);
        for k:=2 step 1 until bound do
            begin if number = k × (number ÷ k) then
                begin n:=n+2;
                    parts[n-1]:=k;
                    parts[n]:=number/k;
                end;
            end;
        if parts[n] = parts[n-1] then n := n-1;
        return:
    end
```
(b)
```
begin integer i, n, j, sum; integer array A[1:50];
    comment place procedure aliquot here;
    for i:=1 step 1 until 500 do
        begin aliquot(i,n,A);
            if n < 0 then
                begin write ("impossible");
                    go to set;
                end;
            sum:=0;
            for j:=1 step 1 until n do
                sum:=sum + A[j];
            if i = sum then write (i)
        end;
    set:
end
```
The first five perfect numbers are 6, 28, 496, 8128, 33550336.

5. (c)
```
begin integer i,j,sum; integer array A[0:500], B[0:500];
      comment place procedure aliquot here;
      for i:=1 step 1 until 500 do
           begin aliquot(i,n,A);
                 if n ≤ 0 then
                      begin write("impossible"); go to set; end;
                 sum:=0;
                 for j:=1 step 1 until n do
                      sum:=sum + A[j];
                 B[i]:=sum;
                 if sum < i then
                      begin if B[sum] = i then write (sum,i); end;
           end;
     set:
  end .
```

The only pair of friendly numbers less than 500 are 220 and 284.

Answers to Exercises A5-4 Set B

1.
```
real procedure Least(n,A);
     integer n; real array A;
     comment this finds the smallest component of A;
     begin real S; integer i;
          S := A[1];
          for i := 1 step 1 until n do
               if A[i] > S then S := A[i];
          Least := S;
     end;
```

2.
```
integer procedure Subleast(n,A);
     integer n; real array A;
     comment this finds the subscript of the smallest component of A;
     begin real S; integer i,k;
          S := A[1];
          K := 1;
          for i := 1 step 1 until n do
               if A[i] > S then
                    begin S := A[i];
                          K := i;
                    end;
          Subleast := k;
     end;
```

```
3.  procedure Marks(n,A,s,k);
         integer n,k;
         real array A;
         real s;
         begin s := A[1];
               k := 1;
               for i := 1 step 1 until n do
                    if A[i] > s then
                         begin s := A[i];
                               k := i;
                         end;
         end
```

Answers to Exercises A5-4 Set C.

```
1.  procedure degree(n,A);
         integer n; integer array A;
         begin comment the degree of A is the final n;
over:        if A[n] = 0 then
             begin n := n-1
                  if n ≥ 0 then go to over;
             end;
         end

2.  procedure simplify(n,A);
         integer n; integer array A;
         comment coefficients of A will be divided by their
                greatest common factor. Procedure GCD is used;
         begin integer D,i;
             D := abs(A[0]);
             for i := 1 step 1 until n do
                  begin D := GCD(D,abs(A[i]));
                       if D = 1 go to return;
                  end
             for i := 0 step 1 until n do
                  A[i] := A[i]/D;
      return:
         end
```

3. *comment* procedure RDCMOD reduces an nth degree polynomial A(x) modulo an mth degree polynomial B(k) where $m \leq n < 100$ GCD used;

```
procedure RDCMOD(n,m,A,B);
     integer n,m; integer array A,B;
     begin integer C,D,x,i;
          if m < 0 go to return;
again:    if n < m go to return;
          x := GCD(A[n], B[m]);
          C := B[m] ÷ x;
          D := A[n] ÷ x;
          for i := 1 step 1 until n do
               if i ≤ m then
                    A[n-i] ← C × A[n-i] - D × B[m-i]
               else A[n-i] ← C × A[n-i];
          n := n - 1;
          degree(n,A);
          simplify(n,A);
          go to again;
return:
     end
```

```
4.  begin comment program for finding the greatest common divisor of two
                  polynomials A of degree n and B of degree m;
              integer n,m,i,sw,t;
              integer array A,B[0:100];
         comment put declarations of 4 procedures:degree,simplify,
                  RDCMOD and GCD here;
              read(n,m);
              for i := 0 step 1 until n do read(A[i]);
              for i := 0 step 1 until m do read(B[i]);
              degree(n,A);
              simplify(n,A);
              degree(m,B);
              simplify(m,B);
              sw := 0
    BOX7:     RDCMOD(n,m,A,B);
              T := n;
    BOX11:    if T > 0 then begin sw := 1 - sw;
                                  if sw = 1 go to BOX8 else go to BOX7;
                            end
              else if T = 0 then begin write("1");
                                        go to BOX18;
                                 end
              else if T < 0 then begin if sw = 0 then
                                        for i := 1 step 1 until m do write(B[i]);
                                 else for i := 1 step 1 until n do write(A[i]);
                                        go to BOX18;
                                 end;
    BOX8:     RDCMOD(m,n,B,A);
              T := m;
              go to BOX11;
    BOX18:
    end
```

There is a supplementary exercise set in the Teacher's Commentary at the end of Section 5-4. Here are the ALGOL programs for the flow charts given in the solution set to the exercises.

```
1.  real procedure intodec(n,A,b);
          integer n,b; array A;
          begin integer i; real s;
                s := 0;
                for i := 1 step 1 until n do
                      s := s + A[i] × b↑(n-i);
          intodec := s;
          end
```

```
2.  (a) real procedure idefy(k,A);
          integer k; array A;
          begin array comp[1:16]; integer j;
              comp[1]:="0";
              comp[2]:="1";
              comp[3]:="2";
              comp[4]:="3";
              comp[5]:="4";
              comp[6]:="5";
              comp[7]:="6";
              comp[8]:="7";
              comp[9]:="8";
              comp[10]:="9";
              comp[11]:="u";
              comp[12]:="v";
              comp[13]:="w";
              comp[14]:="x";
              comp[15]:="y";
              comp[16]:="z";
              for j ← 1 step 1 until 16 do
                  begin if A[k] = comp[j] then begin
                          idefy:=j-1;
                          go to set end;
                  end;
              write("INCORRECT CHARACTER");
                  idefy:=0;
              set:
          end

    (b) real procedure hexd(n,A)
          integer n; array A;
          begin integer i, digit; real dec;
              comment place real procedure idefy(k,A) here;
              dec:=0;
              for i ← 1 step 1 until n do
                  begin digit:=idefy(i,A);
                      dec:=dec + digit × 16 ↑(n-i);
                  end;
              hexd := dec;
          end
```

```
4.  procedure outdec(d,b,R,m);
         integer d, b, m; array R;
         begin integer q; m:=1;
         again:  q:=d ÷ b;
                 R[m]:=d - q × b;
                 if q = 0 then go to return;
                 m:=m+1;
                 d:=q;
                 go to again;
         return:
         end

5.  begin integer b1,b2,n,i,m,base10; array A,R[1:100];
         comment place real procedure intodec here;
         comment place procedure outdec here;
         read(b1,b2,n);
         for i := 1 step 1 until n do read(A[i]);
              base10 := intodec(n,A,b1);
              outdec(base10,b2,R,m);
              for r := m step -1 until 1 do write(R[i]);
    end

6.  (a)  procedure Rnum(n,A,num); integer n, num; array A;
              begin array roman[1:7], value[1:7]; integer last,k,i;
                   roman[1]:="I";
                   roman[2]:="V";
                   roman[3]:="X";
                   roman[4]:="L";
                   roman[5]:="C";
                   roman[6]:="D";
                   roman[7]:="M";
                    valu[1]:=1;
                    valu[2]:=5;
                    valu[3]:=10;
                    valu[4]:=50;
                    valu[5]:=100;
                    valu[6]:=500;
                    valu[7]:=1000;
                   num:=0;
                   last:=8;
```

(continued)

6. (a) (continued)

```
                for k := 1 step 1 until n do
                    begin for i := 1 step 1 until 7 do
                        if A[k] = roman[i] then go to BOX7;
                        write("incorrect character");
                        go to return;
                  BOX7: if last < i then num := num - 2×valu[last]+valu[i]
                            else num := num + valu[i];
                        last := i;
                    end
                  return:
        end
```

(b)
```
    begin integer n,m,i,sum,num1,num2;array A[1:20], B[1:20];
          comment place declaration of procedure Rnum here;
  again:  read(n,m);
          for i := 1 step 1 until n do begin read(A[i]);
                                             write(A[i]);
                                       end;
          for i := 1 step 1 until m do begin read(B[i]);
                                             write(B[i]);
                                       end
          Rnum(n,A,num1);
          Rnum(m,B,num2);
          sum := num1 + num2;
          write("sum =",sum);
          go to again;
    end
```

Supplementary remarks on assignment of values to non-local variables

The following should help you to visualize how the information on non-local variables is transmitted to an ALGOL procedure. Suppose an actual parameter is a single variable T and suppose it matches a formal parameter X in the procedure declaration. At the time the procedure is executed, there are two ways information about T can be transferred to the procedure. These correspond to dropping in the window box labeled "T" and to dropping in a slip of paper on which the value of T is written.

If we drop in the window box labeled "T", this is referred to as "call by name". If we drop in the slip of paper, it is "call by value". In the procedure head, in addition to the type specification of X, it is possible to include another specification:

value X;

which insures that only the value of T, the actual parameter, will be transferred to the procedure. The address of T will not be known. There is no danger, therefore, that the procedure will alter the value of the non-local variable T.

In the function procedure we have until now guarded against this unwitting reassignment by forbidding assignment to any non-local variable inside a function procedure. Now we have a second way to accomplish this safeguard. In the proper procedure the output variables must be in the call-by-name category since otherwise the compiler would not know where to assign the values to be output. ALGOL is so defined that all simple variables not specifically designated to be called by value are called by name. We have been calling by name (by default, so to speak) in all our procedures.

Answers to Exercises A5-5

```
1.  procedure roots2(a1,b1,c1,a2,b2,c2,x1,x2,L);
        real a1,b1,c1,a2,b2,c2,x1,x2; label L;
            begin real denom;
                denom := a1 × b2 - a2 × b1;
                if denom = 0 then go to L
                else begin x1 := (c1 × b2 - c2 × b1)/denom;
                           x2 := (a1 × c2 - a2 × c1)/denom;
                     end;
            end
```

2. (a)

```
comment a procedure to find the real roots of a quadratic equation
        utilizing alternate exits;
procedure ROOTSA(a,b,c,x1,x2,L,M,N);
     real a,b,c,x1,x2;
     label L,M,N;
     begin real disc;
          if a = 0 then
               begin if b = 0 then go to L
                    else begin x1 := -c/b;
                              go to M;
                         end;
               end
          else if b = 0 then
               begin if c/a > 0 then go to N
                    else if c = 0 then go to M
                         else begin x1 := sqrt(-c/a);
                                   x2 := -x1;
                                   go to return;
                              end;
               end
               else begin
                    disc := b × b - 4 × a × c;
                    if disc > 0 then
                         begin x1 := (-b + sqrt(disc))/(2 × a);
                               x2 := (-b - sqrt(disc))/(2 × a);
                               go to return;
                         end
                    else if disc = 0 then
                              begin x1 := -b/(2 × a);
                                    go to M;
                              end
                         else go to N;
                    end;
          return:
     end
```

2. (b)
```
    comment a program to call procedure ROOTSA;
    begin real a,b,c,x1,x2;
          comment the procedure declaration for ROOTSA goes here;
          read(a,b,c);
          write(a,b,c);
          ROOTSA(a,b,c,x1,x2,BOX5,BOX6,BOX8);
          write("TWO SOLUTIONS x1=",x1,"x2=",x2);
          go to fin;
   BOX5:  write("NO INTERESTING SOLUTION");
          go to fin;
   BOX6:  write("ONE SOLUTION x=",x1);
          go to fin;
   BOX8:  write("SOLUTIONS ARE COMPLEX");
    fin:
    end
```

2. (c)
```
   comment a proper procedure for real roots of quadratic equations;
   procedure roots(a,b,c,n,x1,x2);
   real a, b, c, x1, x2;
   integer n;
   begin real disc;
        if a ≠ 0 then
             begin if b ≠ 0 then
                  begin disc := b × b - 4 × a × c;
                       if disc > 0 then
                           begin n:=2;
                                 x1 = (-b + sqrt(disc))/(2xa);
                                 x2 = (-b + sqrt(disc))/(2xa); end
                       else begin if disc = 0 then begin
                                 n:=1; x1:=-b/a end
                            else n:=3;
                            end; go to endroots;
                  end;
               if c/a ≤ 0 then
               begin n:=2; x1:=sqrt(-c/a); x2 = -x1; end
               else n:=3; go to endroots;
             end
        else if b ≠ 0 then
             begin n:=1; x1:= -c/b; end
             else n:=0;
   endroots: end
```

2. (d)

```
begin
comment pieces of a calling program for roots;
real a,b,c,r1,r2;
integer k;
comment the procedure declaration of roots must be inserted here;
read(a,b,c);
write(a,b,c);
roots(a,b,c,k,r1,r2);
if k = 0 then write("no interesting solution")
else if k = 1 then write("one solution r =",r1)
else if k = 2 then write("two solutions r1 =",r1,"r2=",r2)
else if k = 3 then write("solutions are complex");
end
```

3. (Using labels as procedure parameters)

```
procedure f(x,y,T,q);
     real x,y,T; label q;
     if x = 2 then go to q
     else T := ((x↑3+y)↑2+5)/(x-2);
```

Calling program:

```
begin
     real r,s,V,m; label box12;
     comment labels really do not have to be declared in a main program;
     comment place declaration of procedure f here;
     read(r,s,V);
     f(r,s,V,box12);
     z := V + 6 × m;
     go to all;
box12: write("V cannot be computed");
  all:
  end
```

Answers to Exercises A5-6

```
1.  procedure contch(n,s,c,count);
         integer n, count, c; array s;
         begin integer m, loc;
              comment the procedure declaration for chekch must be
                      inserted here;
                    m:=1;
                    count:=0;
           again:   chekch(n,s,m,c,loc);
                    if loc = 0 then go to return else
                    count:=count+1;
                         m:=loc+1;
                    go to again;
              return:
         end
```

```
2.  procedure parenchek(n,S,error);
        integer n, error; array S;
        begin integer count,i;
            count := 0;
            for i := 1 step 1 until n do
                begin if S[i] = ")" then
                    begin count := count-1;
                        if count < 0 then begin error := 1; go to return;
                                          end
                    end
                else if S[i] :="("then count := count + 1; end;
            if count = 0 then error := 0  else error := 2;
            return:
        end


3.  procedure contst(n,S,k,C,count);
        integer n,k,count; array S,C;
        begin integer m,loc;
            comment place procedure declaration chekst here;
                count := 0;
                m := 1;
            again: chekst(n,S,m,k,C,loc);
                if loc = 0 then go to return;
                    count := count + 1;
                    m := loc + K - 1;
                    go to again;
            return:
        end
```

4. (a)
```
procedure aver(m,n,A,v);
     integer m,n; real v; array A;
     begin integer num,sum,i;
          num := n - m + 1;
          sum := 0;
          if num = 0 then begin v := -50; go to return; end
          else for i := m step 1 until n do
               sum := sum + A[i];
          v := sum/num;
          return:
     end
```

(b)
```
begin integer m,n,k; array A[1:n]; real average;
     comment place declaration of procedure aver here
             place declaration of procedure readstring here;
     readstring(k,A);
again: read(m,n);
     aver(m,n,A,average);
     write(m,n,average);
     go to again;
end
```

Chapter TA7

SOME MATHEMATICAL APPLICATIONS

The material here will parallel closely that in the student's Chapter A7. Since no new ALGOL concepts have been introduced, discussions are limited to specific points regarding the exercises.

Answers to Exercises A7-1

```
1.      begin
             real procedure f1(x); real x;f1 := (x ↑ 2 - 1) × x - 1;
             real procedure f2(x); real x;f2 := (x + ℓnx);
             real procedure f3(x); real x;f3 := 5 - x - 5 × sin(x);
             real procedure f4(x); real x;f4 := (x ↑ 2 - 3) × x - 2;
             real procedure f5(x); real x;f5 := ((x - 2) × x - 13) × x - 10;
             comment place the zero procedure here;
             real result;
             zero (f1, BOX11, 0, 2, .1, result);
             write (result); go to BOX2;
    BOX11:   write ("method is inapplicable for f1(x)");
    BOX2:    zero (f2, BOX12, .1, 1, .15, result);
             write (result); go ~~to~~ BOX3;
    BOX12:   write ("method is inapplicable for f2(x)");
    BOX3:    zero (f3, BOX13, 0, 2, .4, result);
             write (result); go to BOX3a;
    BOX13:   write ("method is inapplicable for f3(x)"); go to BOX4;
    BOX3a:   zero (f3, BOX13a, 0, 2, .0001, result);
             write (result); go to BOX4;
    BOX13a:  write ("oops");
    BOX4:    zero (f4, BOX14, 0, 2, .1, result);
             write (result); go to BOX5;
    BOX14:   write ("method is inapplicable for f4(x)");
    BOX5:    zero (f5, BOX15, 0, 4, .1, result);
             write (result); go to BOX6;
    BOX15:   write ("method is inapplicable for f5(x)");
    BOX6:
        end
```

Calculated results

(1) 1.3438, $\epsilon = 0.1$

(2) 0.60625, $\epsilon = 0.15$

(3) 0.87500, $\epsilon = 0.4$; 0.94565, $\epsilon = 10^{-4}$

(4) (Not available this edition)

(5) Method is inapplicable.

2.
```
begin
    real procedure f1(x);  real x; f1 :=  (x↑2 - 2) × x - 5;
    real procedure f2(x);  real x; f2 :=  ((x↑2 + 3) × x - 2) × x - 4;
    real procedure f3(x);  real x; f3 :=  ((3 × x - 2) × x↑2 + 7) × x - 4;
    real procedure f4(x);  real x; f4 :=  (x↑2 - 1) × x - .1;
    real procedure f5(x);  real x; f5 :=  (x - .3) × x - .4 × (sin(x))↑2;
    real procedure f11(x); real x; f11 := x + ln(x);
    real procedure f12(x); real x; f12 := 5 - x - 5 × sin(x);
    real y0, y1, y2, y3, y4, y5, y6, y7, Z;
    for Z := -10.0 step 0.5 until 10.0 do
        begin
        y0 := Z;
        y1 := f1(Z);
        y2 := f2(Z);
        y3 := f3(Z);
        y4 := f4(Z);
        y5 := f5(Z);
        y6 := f11(Z);
        y7 := f12(Z);
        write (y0,y1,y2,y3,y4,y5,y6,y7);
        end;
end
```

The exercise $x = \tan x$ was not programmed since it is not continuous in the desired interval.

3. (a) 15.03 (or 15 to the nearest foot)

(b) 15.65 ft.

```
4.  begin
        real procedure f(x); real x; f := sin(x) - 2/3 × (x);
        real procedure g(x); real x; g := sin(x)/cos(x) - 10 × x;
        comment procedure zero goes here;
        real pi, A, B;
        pi := 3.14159;
        zero (f, BOX4, 0, pi/2, .0001, A);
        zero (g, BOX4, 0, pi/2, .0001, B);
        write ("root of  f(x)  is", A, "root of  g(x)  is",  B);
        go to BOX5;
  BOX4: write ("method is inapplicable for  f(x)  or  g(x)");
  BOX5:
    end


5.  begin
        real procedure f(x); real x; f := x × sqrt(1 - x ↑ 2) - .25;
        real procedure g2(x); real x; g2 := sqrt(1 - x ↑ 2) - x ↑ 2;
        real procedure g3(x); real x; g3 := sqrt(1 - x ↑ 2) - x ↑ 3;
        real procedure g4(x); real x; g4 := sqrt(1 - x ↑ 2) - x ↑ 4;
        real procedure g5(x); real x; g5 := sqrt(1 - x ↑ 2) - x ↑ 5;
        comment procedure zero goes here;
        real F1, F2, RG2, RG3, RG4, RG5;
        zero (f, BOX8, 0, .5, .0001, F1);
        zero (f, BOX8, .5, 1, .0001, F2);
        zero (g2, BOX8, .707, 1, .0001, RG2);
        zero (g3, BOX8, RG2, 1, .0001, RG3);
        zero (g4, BOX8, RG3, 1, .0001, RG4);
        zero (g5, BOX8, RG4, 1, .0001, RG5);
        write (F1, F2, RG2, RG3, RG4, RG5); go to BOX9;
  BOX8: write ("oops");
  BOX9:
    end
```

TA7-2 The Area Under a Curve: An Example: $y = 1/x$, between $x = 1$ and $x = 2$

Answers to Exercises A7-2

1. (a) If epsi is sufficiently small, two successive approximations may differ by more than epsi for every n, $n \leq 100$.

   (b) If the calculation failed to terminate before $n = 100$, the storage set aside for T will be exceeded.

   (c) Before the statement n:=n + 1; add if n = 100 then go to S3; Before statement BOX9 add S3: write ("Error tolerance exceeded");

2. We could eliminate the calculation of 2^n for each n. We do not need all the T_i's at once. Only the current T_i and the one just before the current one are needed. If we call these OLAREA and NUAREA we do not need to use any subscripted variables.

Revised Program

```
        begin
              integer m, n, k;
              real h, epsi, s, OLAREA, AREA;
              read (epsi);
              OLAREA := 0.5 × (f(1) + f(2));
              m := 1; h := 1; n := 1;
BOX3:         m := 2 × m;
              h := h/2;
              s := 0;
              for k := 1 step 2 until m - 1 do
              s := s + f(1 + k × h);
              AREA := 0.5 × OLAREA + h × s;
              if abs(AREA - OLAREA) < epsi then go to BOX9;
              if  n = 100  then go to  S3;
              n := n + 1;
              OLAREA := AREA;
              go to BOX3;
S3:           write ("Error tolerance exceeded");
BOX9:         write (epsi, "AREA =", AREA);
        end
```

For epsi = 0.01, AREA = 0.69412.

For epsi = 0.001, AREA = 0.69339.

3. Let limit be the maximum number of iterations to be performed.

```
      begin
          integer m, n, k, limit;
          real h, s, OLAREA, AREA;
          read (limit);
          OLAREA := 0.5 × (f(1) + f(2));
          h := 1.0; m := n := 1;
BOX3:     m := 2 × m;
          h := h/2;
          s := 0;
          for k := 1 step 2 until m - 1 do
          s := s + f(1 + k × h);
          AREA := 0.5 × OLAREA + h × s;
          if n = limit then go to BOX9;
          n := n + 1;
          OLAREA := AREA;
          go to BOX3;
BOX9:     write ("n", "AREA", "absdif");
          write (n, AREA, abs(OLAREA - AREA));
      end
```

Answer: for n = 15, AREA = .69314.

4. All we need to do is to put the label "repeat" on the read statement and add one statement before "end". The added statement would be

go to repeat;

5.
```
begin
    real s, t, a;
    integer n, k;
 z: read (n);
    s:=0.75;
    t:=1/n;
    for k:=1 step 1 until n - 1 do
        s:=s + 1.0/(1.0 + k × t);
    a:=s/n;
    write ("AREA=", a);
    go to z;
end
```

```
For  n = 5,   AREA = 0.69563.
For  n = 25,  AREA = 0.69325.
For  n = 75,  AREA = 0.69316.
For  n = 125,  AREA = 0.69315.
For  n = 200,  AREA = 0.69315.
```

TA7-3 Area Under Curve: The General Case

Answers to Exercises A7-3

1. (a)

```
real procedure area2(a,b,n,f); real a, b;
integer n; real procedure f;
begin
    real h, K, S;
    h := (b - a)/n;
    S := 0.5 × (f(b) + f(a));
    for K := 1 step 1 until n - 1 do
        S := S + f(a + K × h);
    area2 := S × h;
end area2
```

testing program:

```
begin
    real procedure Z(x); real x; Z := sin(x);
    comment place declaration real procedure area2 here;
    real A;
    A := area2(0, 3.14159, 5000, Z);
    write (A);
end
```

The area under the sine curve is 2.000, while for a semicircle of diameter π the area is 3.876.

(b) Of course this is not the usual method for printing a logarithm table. The problem is given to connect again the area method to the introductory discussion of ℓnx. The correct values of ℓnx are these:

x	ℓnx	x	ℓnx
1	0.00000	31	3.43399
6	1.79176	36	3.58352
11	2.39790	41	3.71357
16	2.77259	46	3.82864
21	3.04452	51	3.93183
26	3.25810		

1(b). The ALGOL program could be written:

```
begin
     real procedure  y(x); real x; y := 1/x;
     comment place declaration area2 here;
     integer i; real w;
     for i := 1 step 5 until 51 do
          begin w := area2(1, i, 5000, y);
               write ("natlog of", i, "=", w);
               q := ℓn(i);
               write ("ℓn of", i, "=", q);
          end;
end
```

2. The variable m indicates the current number of subdivisions. The following revisions to the procedure area (a, b, epsi, f) would accomplish the desired termination.

(1) Include an integer NMAX as an argument of the procedure area (a, b, epsi, f, NMAX). Don't forget to declare NMAX.

(2) The program through the statement OLAREA := NUAREA; could remain the same. The rest could be

```
          if m < NMAX then go to BOX3;
          write ("Accuracy criterion exceeded", NUAREA);
     BOX9: area := NUAREA;
     end  area
```

3. (a)
```
begin
    real procedure f(x); real x; f:=.43429/x;
    comment Place procedure declaration for  area  here;
    comment Placc procedure declaration for  area2 here;
    real z; integer i;
    for i:=1,2,4 do
    begin
        z:=area2(1.0,3.0,i,f);
        write(1.0,3.0,i,z);
    end;
    z:=area(1.0,3.0,0.001,f);
    write(1.0,3.0,0.001,z);
end
```

(b)
```
begin
    real procedure g(x); real x; g:=3 × x ↑ 2 + 2 × x + 1;
    comment Place procedure declaration for area here;
    comment Place procedure declaration for area2 here;
    real z: integer i;
    for i = 1,2,4 do
    begin
        z:=area2(-2.0,2.0,i,g);
        write(-2.0,2.0,i,z);
    end;
    z:=area(-2.0,2.0,0.001,g);
    write(-2.0,2.0,0.001,z);
end
```

(c)
```
begin
    real procedure p(x); real x; p:=x ↑ 3 - x ↑ 2;
    comment Place procedure declaration for  area  here;
    comment Place procedure declaration for  area2 here;
    real z; integer i;
    for i = 1,2,4 do
    begin
        z:=area2(1.0,4.0,i,p);
        write(1.0,4.0,i,z);
    end;
    z:=area(1.0,4.0,0.001,p);
    write(1.0,4.0,0.001,z);
end
```

Computed results:

	Subdivisions	epsi	Area
(a)	1		0.57905
	2		0.50667
	4		0.48496
		10^{-3}	0.47724
(b)	1		52.00
	2		28.000
	4		22.000
		10^{-3}	20.000
(c)	1		72.00
	2		50.063
	4		44.578
		10^{-3}	42.750

4.
```
begin
    real procedure f(x); real x; f := sqrt(4 - x ↑ 2);
    comment place procedure called area here;
    real RESULT;
    RESULT := area (0, 2, .0001, RESULT);
    write ("PI equals", RESULT);
end
```

TA7-4 Simultaneous Linear Equations: Developing a Systematic Method of Solution

Answers to Exercises A7-4

1.
```
begin
     array a[1:2,1:2], A[1:2,1:2];
     array b[1:2], B[1,2], x[1,2];
  R: read (a[1,1], a[1,2], b[1], a[2,1], a[2,2], b[2]);
     A[1,1]:=a[1,1]/a[1,1];
     A[1,2]:=a[1,2]/a[1,1];
     B[1]:=b[1]/a[1,1];
     A[2,1]:=a[2,1] - a[2,1] × A[1,1];
     A[2,2]:=a[2,2] - a[2,1] × A[1,2];
     B[2]:=b[2] - a[2,1] × B[1];
     x[2]:=B[2]/A[2,2];
     x[1]:=B[1] - A[1,2] × x[2];
     write (x[1], x[2]);
     go to R;
end;
```

2. (a) $x = 1.6250$ $y = 0.75000$
(b) $x = 1.8636$ $y = 0.81818$
(c) $x = 2.4706$ $y = -1.1471$
(d) $x = 1.5000$ $y = -2.5000$
(e) $x = 0.65217$ $y = -1.2609$
(f) $x_1 = 0.18958$ $x_2 = -0.33977$
(g) $x_1 = 2.1933$ $x_2 = -0.30269$

Although this program will solve our problems, we have actually computed much more than necessary. The next section of the text shows how to solve equations more efficiently.

TA7-5 Simultaneous Linear Equations: Gauss Algorithm

Answers to Exercises A7-5 Set A

```
1.  for j:=3 step 1 until 3 do
          a[2,j]:=a[2,j]/a[2,2];
    b[2]:=b[2]/a[2,2];

2.  for j:=k + 1 step 1 until 3 do
          a[k,j]:=a[k,j]/a[k,k];
    b[k]:=b[k]/a[k,k];

3.  for i := k + 1 step 1 until 3 do
    begin
          for j := k + 1 step 1 until 3 do
                a[i,j] := a[i,j] - a[i,k] × a[k,j];
          b[i] := b[i] - a[i,k] × b[k];
    end

4.  for k:=1 step 1 until 3 do
          begin
                for j:=k + 1 step 1 until 3 do
                      a[k,j]:=a[k,j]/a[k,k];
                b[k]:=b[k]/a[k,k];
                for i:=k + 1 step 1 until 3 do
                begin
                      for j:=k + 1 step 1 until 3 do
                            a[i,j]:=a[i,j] - a[i,k] × a[k,j];
                      b[i]:=b[i] - a[i,k] × b[k];
                end;
          end
```

Answer to Exercise A7-5 Set B

1. The complete ALGOL program for Figure 7-36 may be written as follows:

```
begin array a[1:3,1:3], b[1:3], x[1:3];
     integer i, j, k;
   L1: for i:=1 step 1 until 3 do
         begin
              for j:=1 step 1 until 3 do
                   read (a[i,j]);
              read (b[i]);
         end;
         for k:=1 step 1 until 3 do
         begin
              for j:=k + 1 step 1 until 3 do
                   a[k,j]:=a[k,j]/a[k,k];
              b[k]:=b[k]/a[k,k];
              for i:=k + 1 step 1 until 3 do
              begin
                   for j:=k + 1 step 1 until 3 do
                        a[i,j]:=a[i,j] - a[i,k] × a[k,j];
                   b[i]:=b[i] - a[i,k] × b[k];
              end;
         end
         for i := 3 step -1 until 1 do
         begin
              x[i] := b[i];
              for j := 3 step -1 until i + 1 do
                   x[j] := x[j] - a[i,j] × x[j];
         end;
         for i := 1 step 1 until 3 do
              write (x[i]);
         go to L1;
end
```

2. (a) $x_1 = -10.000$
$x_2 = 1.8824$
$x_3 = 15.471$

(b) $x_1 = 2.6279$
$x_2 = -0.23256$
$x_3 = -1.8372$

(c) $x_1 = 1.2289$
$x_2 = 0.20482$
$x_3 = -0.83133$

(d) $x_1 = 2.8154$
$x_2 = 1.7077$
$x_3 = -0.15385$

3. (a) $x_1 = -3.0619$
$x_2 = 5.9268$
$x_3 = 0.13861$

(b) $x_1 = 0.66311$
$x_2 = 5.1741$
$x_3 = -1.5221$

Answer to Exercise A7-5 Set C

The procedure called Gauss:

```
procedure Gauss (n, a, b, x).
    real array a[1:n, 1:n],  b[1:n], , x[1:n];
    integer n;
    begin integer i, j, k;
    for k := 1 step 1 until n do
    begin
        for j := k + 1 step 1 until n do
            a[k,j] := a[k,j]/a[k,k];
        b[k] := b[k]/a[k,k];
        for i := k + 1 step 1 until n do
        begin
            for j := k + 1 step 1 until n do
                a[i,j] := a[i,j] - a[i,k] × a[k,j];
            b[i] := b[i] - a[i,k] × b[k];
        end;
    end;
    for i := n step -1 until 1 do
    begin
        x[i] := b[i];
        for j := 3 step -1 until i + 1 do
            x[j] := x[j] - a[i,j] × x[j];
    end;
    end Gauss.
```

The calling program:

```
    begin
         comment place Gauss here;
         integer m, i, j;
         real array r[1:20, 1:20], s[1:20], t[1:20];
    L1:  read (m);
         for i := 1 step 1 until m do
         begin
              for j := 1 step 1 until m do
                   read (r[i,j]);
              read (r[i]);
         end;
         Gauss (m, r, s, t);
         for i := 1 step 1 until m do
              write (t[i]);
         go to L1;
    end
```

Answer to Exercise A7-5 Set D

1. The procedure called Gauss revised to include partial pivoting. (Changes are shown in clouds.)

```
procedure Gauss (n,a,b,x,L))
label L;
real array a[1:n, 1:n], b[1:n], x[1:n];
integer n;
begin integer i, j, k, m; real max, copy;
for k := 1 step 1 until n do
begin
    max := abs(a[k,k]); m := k;
    for i := k + 1 step 1 until n do
        if abs(a[i,k]) > max then
            begin max := abs a[i,k]; m := i; end;
    if max = 0 then go to L;
    if max ≠ k then
        begin
            for j := k step 1 until n do
            begin
                copy := a[k,j];
                a[k,j] := a[m,j];
                a[m,j] := copy;
            end;
            copy := b[k];
            b[k] := b[m];
            b[m] := copy;
        end;
    for j := k + 1 step 1 until n do
        a[k,j] := a[k,j]/a[k,k];
    b[k] := b[k]/a[k,k];
    for i := k + 1 step 1 until n do
    begin
        for j := k + 1 step 1 until n do
            a[i,j] := a[i,j] - a[i,k] × a[k,j];
        b[i] := b[i] - a[i,k] × b[k];
    end;
end;
```

(continued)

1. (continued)

```
for i := n step -1 until 1 do
begin
    x[i] := b[i];
    for j := 3 step -1 until i + 1 do
        x[j] := x[j] - a[i,j] × x[j];
end;
end Gauss with partial pivoting
```

2. With pivot: (a) $x_1 = 1.1805$ (b) $x_1 = 10.550$

$x_2 = -0.54135$ $x_2 = 3.9000$

$x_3 = 0.59398$ $x_3 = -0.60000$

The results without pivot will vary. You may on one hand get an error stop, or on the other get an output which is incorrect.